J500

Aplicações Distribuídas com J2EE™ e JBoss™

Revisão 1.5 (Junho 2003)

Helder da Rocha
www.argonavis.com.br

Objetivos

- *Este curso tem como objetivo oferecer uma introdução prática às tecnologias J2EE, mostrando como usar em conjunto as tecnologias*
  - *Enterprise JavaBeans*
  - *Servlets e JSP*
- *Como módulos opcionais também são abordados*
  - *JavaMail API*
  - *XML e Web Services com JAX-RPC*
- *Para atingir esses objetivos, usaremos apenas recursos padrão e ferramentas de código aberto*
  - *JBoss application server para produção*
  - *Apache Ant para o ambiente de desenvolvimento*

Pré-requisito fundamental

Experiência com programação em Java

Assuntos abordados

- Este curso explora os seguintes assuntos
  - Como escrever componentes J2EE (Web e EJB), como compilá-los, empacotá-los, implantá-los e executá-los:
  - Como criar Session beans, Entity beans, Message-driven beans, servlets e JSP
  - Como escrever aplicações distribuídas compostas de componentes Web e EJB combinados
  - Como configurar componentes J2EE para usar serviços disponibilizados pelo servidor de aplicações de maneira declarativa
  - Uso de JNDI para acesso ao serviço de nomes
  - Uso de ambiente de desenvolvimento J2EE baseado no Ant

Assuntos abordados superficialmente

- Os seguintes assuntos são abordados de maneira superficial neste curso
  - EJB-QL e relacionamentos entre EJBs
  - APIs para acesso programático acesso a serviços (transações, autenticação e persistência): BMP, JTA, JAAS
  - Autenticação de usuários e autorização para acesso a recursos no servidor
  - Boas práticas e padrões de projeto J2EE (discutidos superficialmente nos módulos finais de cada seção)
  - MVC e modelagem de aplicações Web

Assuntos não abordados

- Os seguintes assuntos não são abordados neste curso
  - Java e JDBC básico (pré-requisitos)
  - Configuração de autenticação e autorização em servlets e JSP (controlamos apenas métodos de negócio em EJBs)
  - Tópicos avançados de transações ou segurança
  - Tópicos avançados de arquitetura de aplicações J2EE: modelagem, relacionamentos avançados com EJB-QL
  - Análise, design ou técnicas e práticas para desenvolver aplicações robustas, seguras e eficientes
  - Padrões de projeto J2EE
  - UML (Unified Modelling Language)
  - Uso de ambientes integrados (IDEs) gráficos, debuggers, profilers, sistemas de controle de versão, etc.

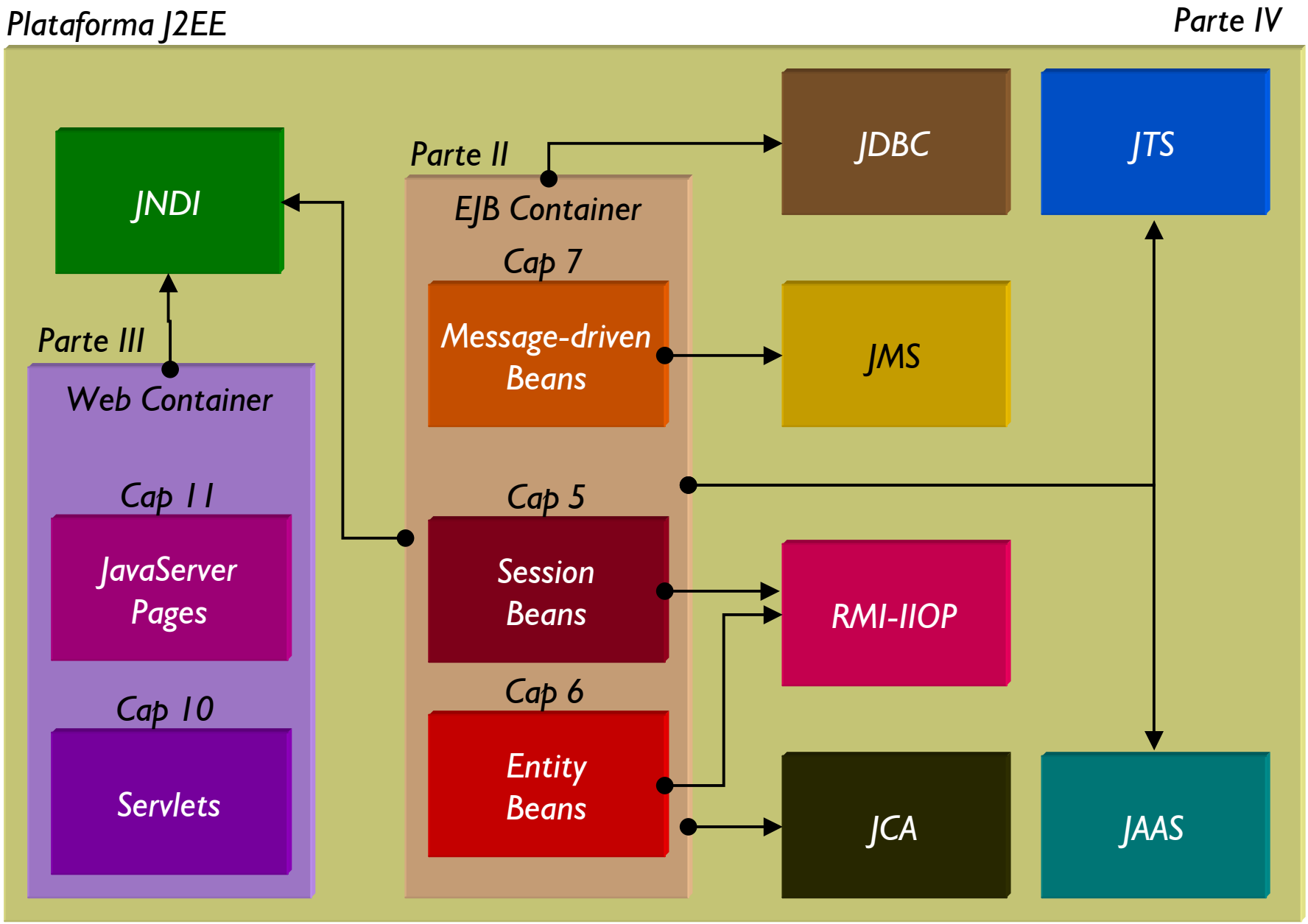
Roteiro (1)

- Parte I Fundamentos
  - 1. Plataforma J2EE: introdução prática
  - 2. Ambiente de desenvolvimento: Apache Ant
  - 3. Serviços de Localização (JNDI)
- Parte II Tecnologia EJB
  - 4. RMI-IIOP e Enterprise JavaBeans
  - 5. Session Beans
  - 6. Entity Beans (usando CMP)
  - 7. JMS e Message-driven Beans
  - 8. Construção de Aplicações EJB

Roteiro (II)

- *Parte III Tecnologia Web*
  - *9. Introdução à aplicações Web*
  - *10. Servlets*
  - *11. JavaServer Pages*
  - *12. Construção de aplicações Web*
- *Parte IV Aplicações J2EE*
  - *13. Recursos compartilhados e Transações*
  - *14. Autenticação e Autorização*
  - *15. Construção de uma aplicação J2EE*
- *Módulos extras*
  - *Java Mail API*
  - *XML e Web Services com JAX-RPC*

Plataforma J2EE e assuntos abordados



Resultados esperados

- *Ao final deste curso você terá os conhecimentos essenciais da arquitetura J2EE e saberá como implantar aplicações distribuídas em qualquer servidor compatível. Será capaz de*
  - *Criar componentes Enterprise JavaBeans*
  - *Criar servlets e páginas JSP para aplicações Web;*
  - *Empacotar aplicações em componentes JAR, WAR e EAR;*
  - *Instalar, configurar, implantar e usar (na implementação de referência e no JBoss) aplicações distribuídas consistindo de enterprise beans, servlets e páginas JSP;*
  - *Construir aplicações onde servlets, páginas JSP, session beans, entity beans e message-driven beans comunicam-se entre si*
  - *Controlar, em cada componente, o acesso aos serviços de autenticação, persistência e transações do servidor de forma declarativa*

Como tirar o melhor proveito deste curso

- *Faça perguntas*
- *Faça os exercícios*
- *Explore os exemplos*
- *Vá além dos exemplos e exercícios: invente exemplos similares, teste trechos de código*
- *Explore e se familiarize com a documentação*
- *Procure desenvolver um projeto que utilize Java, seja no trabalho, seja no seu tempo livre*
  - *Não fique sem programar nos próximos meses ou todo o esforço terá sido em vão!*
- *Leia revistas, artigos e livros sobre Java e J2EE e mantenha-se atualizado.*

Recursos didáticos utilizados

- Apresentação
  - Slides (em alguns módulos) e demonstrações interativas
    Roteiros didáticos em livros-texto (veja a seguir)
- Exercícios são propostos ao final de cada módulo
  - Aplicações triviais (para fixar conceitos aprendidos)
- Projetos J2EE
  - No final de cada parte há um módulo onde a tecnologia abordada é explorada de forma prática, abrindo espaço para discussão sobre arquitetura, padrões e práticas
  - No final da parte I é construída uma aplicação EJB empregando vários EJBs em conjunto
  - No final da parte II é construída uma aplicação Web
  - Na parte III as duas aplicações são combinadas

CD

- *O CD que acompanha este curso contém todo o material necessário, software e fontes adicionais de informação*
- *A sua estrutura geral é a seguinte*
  - ***cap01** a **cap15**: arquivos com código-fonte correspondente a cada módulo do curso. Cada um possui um ou mais build.xml (roteiro para o Ant) que permite instalar e rodar as aplicações*
  - ***slides**: contém as apresentações em PDF utilizadas em aula*
  - ***software**: contém todo o software usado em aula e mais (J2SDK, J2EE SDK Win e Linux, JBoss, SunONE App Server, OpenLDAP, Joram, Eclipse, JEdit, JUnit, Cactus, HttpUnit, etc.)*
  - ***docs**: tutoriais online da Sun, especificações J2EE e CORBA em PDF, whitepapers, código-fonte*
- *O objetivo do CD é facilitar o acesso ao material do curso. Sempre que possível, procure versões mais atuais na Internet.*

Fontes

- *Este material (slides) é utilizado como roteiro de treinamento. Não pretende ser uma fonte detalhada de informação sobre J2EE. Para detalhes, consulte as fontes abaixo*

Veja no CD em docs/

- [1] *The J2EE Tutorial, Sun*
  http://java.sun.com/j2ee/tutorial/
- [2] *Mastering EJB 2.0, Ed Roman et al. e EJB Design Patterns, F. Marinescu*
  http://www.theserverside.com/books/masteringEJB/index.jsp
  http://www.theserverside.com/books/EJBDesignPatterns/index.jsp
- [3] *The JNDI Tutorial, Sun*
  http://java.sun.com/products/jndi/tutorial/
- [4] *The JMS Tutorial, Sun*
  http://java.sun.com/products/jms/tutorial/
- [5] *The Java Web Services Tutorial, Sun*
  http://java.sun.com/webservices/docs/ea2/tutorial/index.html
- [6] *Java and XML, 2ed, Bret McLaughin (publicado pela O'Reilly)*
- [7] *Enterprise JavaBeans, 3ed, Richard Monson-Haefel (O'Reilly)*
- [8] *Web Development with JavaServer Pages, D. Fields, M. Kolb (Manning)*

livros-texto principais

livros-texto usados em módulos específicos

Check-list de instalação em Windows

- J2SDK 1.4.2
  - Rode /software/java/j2sdk1.4.2-win.exe
- J2SDK EE 1.4 Beta 2
  - Rode /software/j2ee/j2sdkee-1_4Beta2-win.exe
- JBoss 3.2.1
  - Descompacte o arquivo /software/j2ee/jboss-3.2.1.zip em C:\
- Jakarta Ant
  - Descompacte /software/java/jakarta-ant-1.5-bin.zip em C:\
  - Mude o nome do diretório raiz criado (jakarta-ant-*) para ant
- JEdit 4.1
  - Rode /software/java/jedit41install.jar e siga as instruções

Check-list de configuração (Windows)

- Crie as seguintes variáveis de ambiente
  - JAVA_HOME=c:\j2sdk1.4.2
  - ANT_HOME=c:\ant
  - J2EE_HOME=c:\j2sdkee1.4Beta2
  - JBOSS_HOME=c:\jboss-3.2.1
- Acrescente, à sua variável PATH os seguintes caminhos
  - %JAVA_HOME%\bin;%J2EE_HOME%\bin;%ANT_HOME%\bin;
- Crie atalhos na sua área de trabalho para:
  - c:\jboss-3.2.1\bin\run.bat

helder@argonavis.com.br

argonavis.com.br

Plataforma J2EE: fundamentos e introdução prática

Helder da Rocha
www.argonavis.com.br

Objetivos

- Este módulo tem três objetivos
  - Configurar o ambiente de trabalho
  - Oferecer uma visão geral da tecnologia J2EE definindo os seus conceitos fundamentais e mostrando quais os problemas que a tecnologia pretende solucionar
  - Apresentar uma introdução prática à tecnologia J2EE através do desenvolvimento, montagem, instalação e configuração de componentes simples
- Para a parte prática usaremos
  - Exemplo do capítulo 3 do livro Mastering EJB2*
  - Roteiro e exemplos do capítulo 1 do livro J2EE Tutorial* (alterados para implantação no JBoss)
  - Veja diretório cap01 no CD deste curso

* Veja referências no final

Sumário

- Introdução: ambiente e arquivos
- Conceitos fundamentais
  - Servidores de aplicação
  - Arquitetura de componentes
  - Plataforma J2EE: arquitetura, APIs, serviços e papéis
  - Componentes J2EE
- Infraestrutura
  - Ferramentas do J2EE SDK: deploytool, servidor e verifier
  - Servidor JBoss
- Introdução prática
  - Tecnologia EJB: como criar e instalar um Session Bean
  - Tecnologia Web: como criar e instalar um JSP (WAR)
  - Implantação de um Enterprise Archive (EAR)

Estrutura dos exemplos no CD

- *Cada módulo possui um diretório no CD que contém arquivos com exemplos. Para a maioria, a compilação, montagem e execução pode ser feita através do Ant*
  - *O Ant executa um roteiro escrito em XML: o buildfile (`build.xml`)*
  - *Entender a linguagem do Ant ajuda a entender como as aplicações são montadas e executadas*
- *Estrutura típica da maior parte dos diretórios de exemplos*

`src/`	*Contém os fontes*
`web/`	*Contém as fontes Web (HTML, JSP, GIFs, JPG*
`etc/` *ou* `dd/`	*Contém deployment descriptors (XML)*
`lib/`	*Contém bibliotecas requeridas*
`build.xml`	*Buildfile usado pelo Ant (alguns requerem modificação)*
`build.properties`	*Configure com os dados do seu ambiente!*

- *Principais diretórios gerados pelo Ant (pode haver outros)*

`build`	*ou* `classes` *contém código compilado*
`release`	*contém componentes empacotados em JARs, WARs e EARs*

Tecnologias "corporativas"

- "Enterprise" é o nome usado para identificar as tecnologias Java que são usadas em sistemas distribuídos
  - "Enterprise" == "Distributed"
- São várias as tecnologias Java para sistemas distribuídos
  - Banco de dados: JDBC, SQLJ, JDO
  - Objetos distribuídos: Java RMI, RMI-IIOP, Java IDL (CORBA), EJB
  - Serviços distribuídos: JTA, JTS, JAAS, DGC, JNDI
  - Eventos distribuídos: JMS
  - Web: Servlets, JSP, JSTL
  - XML e Web services: JAXP, JAXB, JAXM, JAXR, JAX-RPC
  - E-mail: JavaMail
  - ...
- Algumas tecnologias podem ser usadas através de APIs do J2SE. Outras são distribuídas apenas no pacote J2EE.

Banco de dados

- *JDBC é a API padrão*
  - *Pacote java.sql*
  - *Conecta programas em Java, de forma transparente, a bancos de dados relacionais*
  - *Maior parte dos bancos do mercado possuem drivers*
- *Extensões opcionais*
  - *Pacote javax.sql*
  - *Fazem parte de J2EE*
  - *Suporte a RowSets e DataSources*
  - *javax.sql.DataSource permite a obtenção de uma conexão de um pool de conexões via JNDI*

```
DataSource ds = (DataSource) ctx.lookup("jdbc/Banco");
Connection con = ds.getConnection();
```

Objetos distribuídos

- *Java RMI (sobre JRMP)*
  - *Solução 100% Java para comunicação em rede*
  - *Muito mais simples e fácil de manter e expandir que soluções baseadas em sockets*
- *Java IDL*
  - *Implementação de CORBA em Java*
  - *Solução para integração com aplicações escritas em outras linguagens*
- *RMI sobre IIOP*
  - *Modelo de programação RMI com execução CORBA*
- *EJB*
  - *Arquitetura de componentes baseada em RMI sobre IIOP*

Serviços distribuídos

- *JTS (Java Transaction Service)*
  - *Serviço de transações distribuído compatível com CORBA Transaction Service (OTS)*
- *JTA (Java Transaction API)*
  - *API de programação Java que implementa o padrão XA do Open Group para transações distribuídas (two phase commit protocol)*
- *DGC (Distributed Garbage Collection)*
  - *Remove referências remotas que não podem mais ser usadas*
  - *Disponível apenas em aplicações Java RMI*
- *JNDI (Java Naming and Directory Interface)*
  - *Interface padrão genérica para sistemas de nomes*
- *JAAS (Java Authorization and Authentication Services)*
  - *API para implementação de sistemas de login com reconhecimento de identidade do usuário.*

Eventos distribuídos

- JMS (Java Message Service)
  - "Message" == "Event" em ambientes distribuídos
  - API que permite interagir com diferentes sistemas de Message Oriented Middleware (MOM)
  - Permite criar sistemas distribuídos com comunicação assíncrona
  - Alternativa aos protocolos síncronos (IIOP e RMI/JRMP)

Serviços Internet (Web e email)

- Componentes Web
  - Servlets: substitui CGI, ISAPI, e similares estendendo serviço HTTP
  - JSP: modelo de programação orientado à interface para servlets; substitui ASP, PHP, CFML e similares
- Web Services
  - JAXP: coleção de APIs para leitura, geração e transformação de XML. Suporte a DOM, SAX e XSLT
  - JAXB: API para mapeamento Java-XML. Usada para gerar código Java a partir de documentos XML
  - JAXM: API para troca de mensagens XML com SOAP
  - JAX-RPC: API para implementar aplicações RPC com SOAP
  - JAXR: API para acesso a registros UDDI, ebXML
- JavaMail
  - API para a programação de clientes SMTP e IMAP

Servidores de aplicação

- Servidores de aplicação OO, também chamados de Component Transaction Monitors, oferecem ambientes para operação de componentes (rodando em containers) e diversos serviços de middleware
- Servidores que suportam EJB permitem oferecem vários serviços de middleware implícito*, entre eles
  - Controle de transações
  - Autenticação e autorização
  - Gerenciamento de recursos
  - Persistência
- Nos servidores EJB esses serviços são configurados através de arquivos XML
- Os serviços também podem ser usados de forma explícita*

* mais detalhes no capítulo 9

Java 2 Enterprise Edition

- J2EE é
  - Uma especificação para servidores de aplicação que define padrão de suporte a componentes e serviços
  - Um pacote de APIs e ferramentas para desenvolver componentes que rodam nesses servidores
- É objetivo da plataforma J2EE reduzir o custo e a complexidade de desenvolver serviços multi-camada
- Servidores de aplicação compatíveis com a especificação J2EE oferecem
  - Suporte à arquitetura de componentes EJB
  - Suporte a serviços Web, servlets e JSP
  - Suporte a serviços de middleware explícito e implícito

Componentes J2EE

- Aplicações J2EE são aplicações em n-camadas feitas de componentes
- A especificação J2EE define os seguintes componentes
  - Componentes cliente - rodam na máquina do usuário
  - Componentes Web (JSP e servlets) - rodam em servidor
  - Componentes EJB - rodam em servidor
- Componentes J2EE são escritos em Java
  - Aderem a um determinado formato padrão de construção definida na especificação J2EE que inclui regras para a construção de métodos, classes, arquivos XML de configuração e empacotamento
  - Implantáveis (deployable) para produção em servidores de aplicação compatíveis com a plataforma J2EE

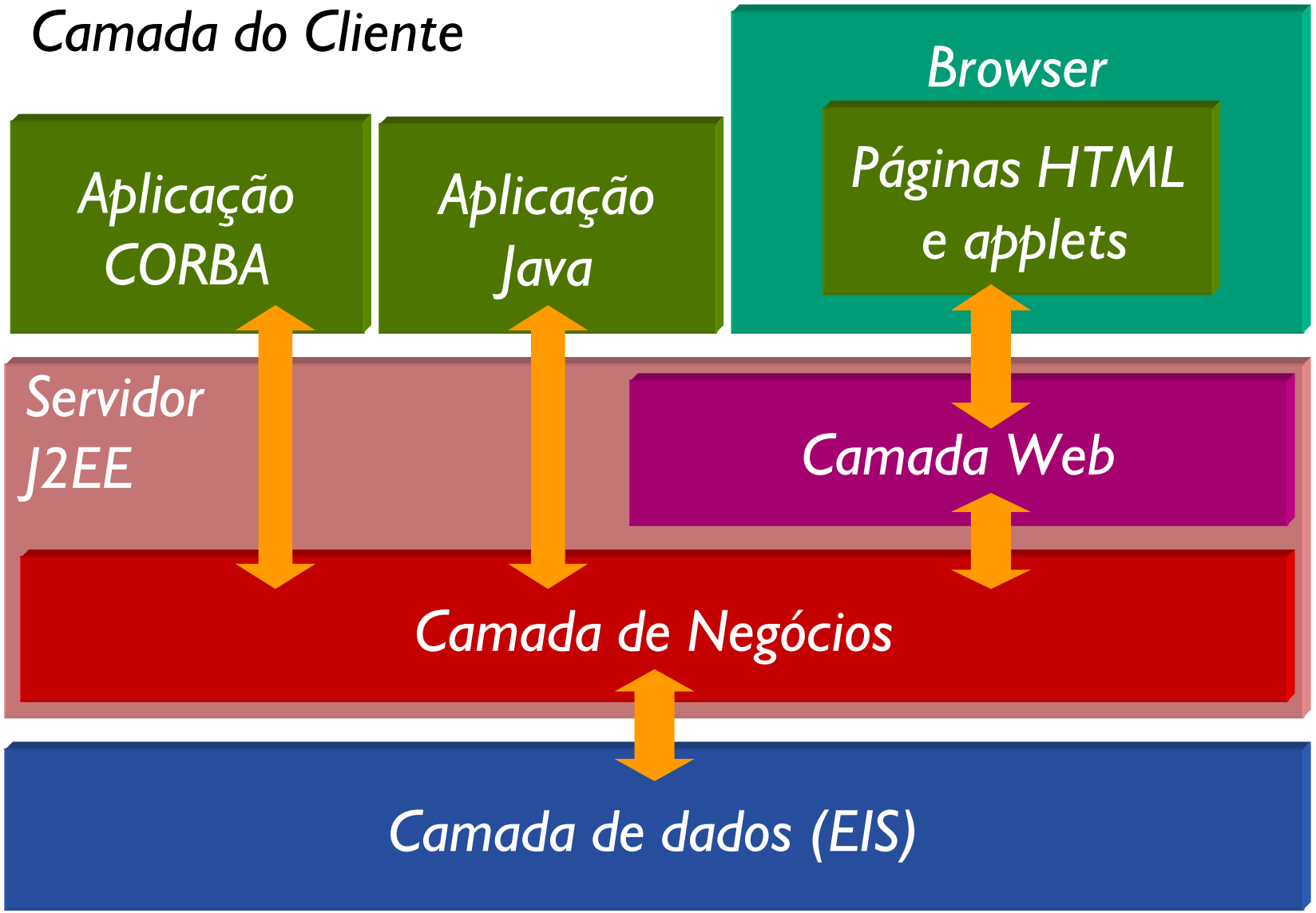
Arquitetura de componentes

- *É um contrato entre o fornecedor e o usuário*
  - *Permitem que os fornecedores construam componentes compatíveis com uma classe de servidores: reuso!*
  - *Permite que os usuários substituam componentes existentes por outros similares: flexibilidade!*
- *Servidores de aplicações são usuários de componentes*
  - *Cada servidor utiliza componentes que obedecem à determinada arquitetura de componentes*
- *Plug & Play*
  - *Um componente pode ser "plugado" no servidor de aplicações e passar a funcionar imediatamente.*
- *Analogia*
  - *Se CD-ROM é servidor de aplicações, o CD é componente*

Containers J2EE

- *Um container é a interface entre o componente e as funções de baixo nível da plataforma onde roda*
  - *Antes que um componente EJB ou Web possa ser executado em um container J2EE ele precisa ser montado em uma aplicação J2EE e implantado no seu container*
- *O container é responsável por chamar os métodos que controlam o ciclo de vida dos componentes*
- *O container também é quem serve de interface para que o componente utiliza serviços de middleware implícito declarados nos seus arquivos de configuração*
- *A plataforma J2EE oferece três tipos de containers*
  - *Container EJB*
  - *Container Web*
  - *Container Cliente (para executar aplicações cliente)*

Arquitetura em Camadas



Componentes da camada do cliente

- Clientes Web (browsers ou web services)
  - Acesso indireto a EJBs via camada Web
  - Mídia Web estática (HTML, XML, GIF, etc.) geradas por componentes Web no servidor
  - Clientes sempre magros (não contém lógica de negócio)
- Clientes de aplicação
  - Podem ter acesso direto a EJBs na camada de negócios
  - Consistem de aplicações de linha de comando, aplicações gráficas AWT ou Swing e permitem uma interface do usuário mais sofisticada
- Outros clientes
  - Applets - podem ser clientes Web ou de aplicação
  - Clientes CORBA - acesso direto a EJBs via IIOP

Componentes da camada Web

- Servlets
  - Classes pré-compiladas que processam requisições HTTP e devolvem respostas HTTP de qualquer tipo
  - Ideais para a geração de conteúdo dinâmico que não é enviado para o browser como texto (imagens, vídeos, arquivos ZIP, Flash, etc.)
  - Usados como controladores em aplicações JSP
- JavaServer Pages (JSP)
  - Páginas de texto contendo Java embutido que operam como servlets
  - Compiladas após a instalação ou durante a execução
  - Ideais para gerar páginas de texto, HTML e XML (porta de comunicação para Web Services)

Componentes da camada de negócio

- Enterprise JavaBeans (EJB)
  - Formam o núcleo de uma aplicação distribuída
  - Recebem e processam dados de clientes e enviam (transparentemente) à camada de armazenamento
  - Recuperam dados da camada de dados, processam e enviam para clientes
- Enterprise JavaBeans são objetos CORBA*
  - Acessíveis via IIOP**, podem ser chamados por clientes CORBA (mesmo clientes escritos em outras linguagens)
- EJBs sempre são escritos em Java
  - São desenvolvidos usando RMI sobre IIOP: modelo de programação Java que gera objetos CORBA

* Common Object Request Broker Architecture - padrão OMG para objetos distribuídos

** Internet Inter-ORB Protocol (mais sobre isto no módulo 3)

Três tipos de enterprise beans

- Session Beans
  - Modelam processos de negócio. São ações, verbos.
  - Fazem coisas: acessam um banco, fazem contas,
  - Podem manter ou não estado não-persistente
  - Processar informação, comprar produto, validar cartão
- Entity Beans
  - Modelam dados de negócio. São coisas, substantivos.
  - Representam informações em bancos de dados
  - Mantêm estado persistente
  - Um produto, um empregado, um pedido
- Message-driven beans
  - Modelam processos assíncronos. Respondem a eventos.
  - Agem somente quando recebem uma mensagem
  - Não mantêm estado

Enterprise JavaBeans vs. JavaBeans

- Um Enterprise JavaBean não é um tipo de JavaBean
- Ambos fazem parte de uma arquitetura de componentes
  - Implementam um contrato que permite o seu reuso por alguma outra aplicação padrão ou framework
- A arquitetura de componentes Enterprise JavaBeans define
  - Regras para construir componentes contendo classes, interfaces e arquivos XML de configuração visando a implantação automática em servidores EJB
  - Um EJB consiste de no mínimo três classes e um XML em um JAR
- A arquitetura de componentes JavaBeans define
  - Regras para construir classes, para permitir um tratamento especial por parte de ferramentas e frameworks
  - Um JavaBean consiste de no mínimo uma classe contendo um método get() e um construtor sem argumentos
- JavaBeans não são parte da arquitetura J2EE

Papéis definidos na especificação J2EE

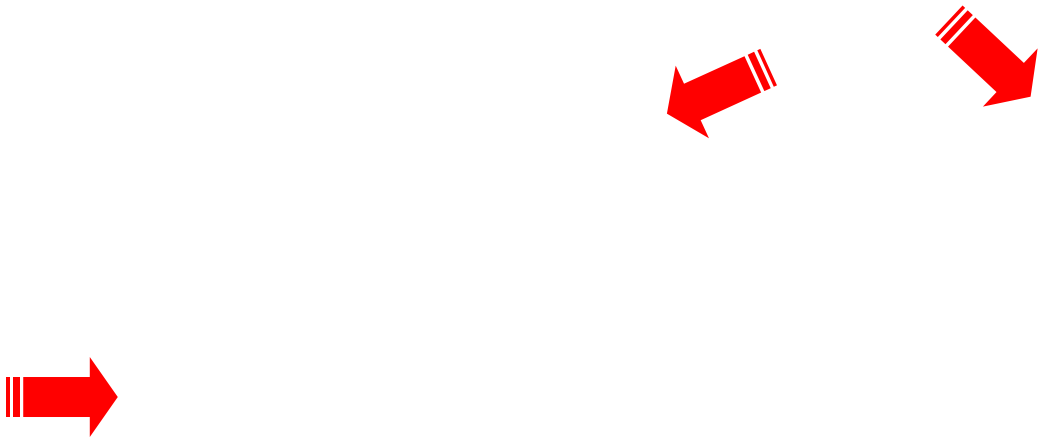
- Provedor de componentes (bean provider)
  - Desenvovedor que cria os componentes J2EE
- Provedor de ferramentas (tool provider)
  - Oferecem ferramentas para facilitar a montagem e manutenção das aplicações J2EE
  - Ferramentas de modelagem de dados, ferramentas de montagem de componentes, etc.
- Montador de aplicações (application assembler)
  - Arquiteto de aplicações que monta aplicações usando componentes desenvolvidos pelo provedor de componentes com as ferramentas do provedor de ferramentas
  - Escreve código de integração entre componentes, lógica de relacionamentos, etc.

Papéis J2EE (2)

- Provedor dos containers (container/server provider)
  - Fabricantes que oferecem containers EJB e Web
  - Garantem o ambiente de execução onde vivem os beans e os serviços de middleware que eles têm acesso
  - JBoss, WebLogic, WebSphere, Tomcat (Web), IPlanet, etc.
- Implantador de aplicações (deployer)
  - Instala e coloca para funcionar a aplicação no software fornecido pelo provedor de containers
  - Garante a integração com sistemas e serviços, escolhe o hardware, ajusta a segurança, performance, acesso a serviços e recursos externos
- Administrador do sistema (system administrator)
  - Garante a estabilidade da solução operacional

Por que separação de papéis?

- Camadas distintas (devido à arquitetura J2EE) permitem que companhias ou indivíduos se especializem em certos papéis
- Alguns papéis podem ser combinados
  - Em pequena empresa, bean provider, application assembler e deployer podem ser mesma pessoa



Em destaque: principais pápéis relevantes para este curso

Fonte da ilustração: Ed Roman et al. Mastering EJB 2, pag. 21

API J2EE (pacotes top-level)

Disponíveis em j2ee.jar

- *javax.activation* — JavaBeans Activation Framework usado pelo JavaMail
- *javax.ejb* — Classes e interfaces para construir EJBs
- *javax.jms* — Classes e interfaces para construir aplicações JMS
- *javax.mail* — Classes que modelam um cliente de e-mail
- *javax.resource* — JCA: API para desenvolvimento de conectores (RARs)
- *javax.security.auth* — JAAS: API de autenticação e autorização
- *javax.servlet* — Classes e interfaces para construir servlets e páginas JSP
- *javax.sql* — Pacote JDBC estendido
- *javax.transaction* — JTA: API para controle de transações
- *javax.xml.parsers* — JAXP: Classes para processamento XML
- *javax.xml.transform* — Classes para processamento de transformações XSLT
- *org.w3c.dom* — Implementação de DOM, componente do JAXP.
- *org.xml.sax* — Implementação de SAX, componente do JAXP

Componentes EJB e EJB-JAR

- Cada **componente EJB** contém
  - Um Enterprise Bean (*javax.ejb.EnterpriseBean*)
    - Implementa a lógica da aplicação em uma das três sub-interfaces de EnterpriseBean: EntityBean, SessionBean ou MessageDrivenBean
  - Uma Interface Home (*javax.ejb.EJBHome*)
    - Fábrica usada pelos clientes para obter instâncias do EJB
  - Uma Interface Remote (*javax.ejb.EJBObject*) e/ou
    - Declara os métodos da interface de negócio de cada bean
  - Interfaces locais (*javax.ejb.EJBLocalObject* e *EJBLocalHome*)
    - Alternativas às interface Home e Remote para acesso local eficiente
  - Uma entrada no Deployment Descriptor (*ejb-jar.xml*)
    - Arquivo XML que associa cada uma das classes acima com seu enterprise bean e permite definir e configurar o uso de serviços como transações, persistência, permissões de acesso, referências, etc.
- As classes, com o ejb-jar.xml são guardadas em um JAR

Web Application Archive e Client-JAR

- **Componentes Web** armazenados em arquivos WAR, contém
  - Páginas JSP, páginas HTML, imagens, arquivos Flash, Applets
  - Servlets, JavaBeans e outras classes
  - JARs contendo classes usadas pelos componentes
  - Bibliotecas de tags para JSP (Taglibs)
  - Tag library descriptors (*.tld) para cada biblioteca incluída
  - Web Deployment Descriptor (web.xml), que descreve os mapeamentos entre servlets e nomes de contexto, filtros, parâmetros iniciais, referências com EJBs e outras informações
- **Componentes Cliente*** são armazenados em arquivos JAR
  - Application Client Descriptor (application-client.xml), armazenado junto com as classes cliente descrevem as referências para EJBs chamados, associando-os a nomes JNDI no domínio java:comp/env

* clientes EJB e clientes Web não precisam ser "componentes J2EE" (que rodam em container)
Se um servidor oferece um container para aplicações cliente, o desenvolvimento do cliente como componente será facilitado pois o mesmo poderá usar serviços do seu container

Enterprise Archive

- EJB-JARs, JARs de componente-cliente e WARs podem ser empacotados em JARs chamados de arquivos **EAR**
- Um arquivo EAR deve conter
  - Os JARs, WARs e EJB-JARs da aplicação
  - Bibliotecas adicionais opcionais
  - Um Application Deployment Descriptor (application.xml), que descreve cada módulo (contém o nome do JAR, WAR ou EJB-JAR dentro de um tag que descreve seu tipo)
- Além desses arquivos EJB-JARs, WARs e EARs podem conter arquivos dependentes do fabricante

Arquivos dependentes do fabricante

- O fabricante de um servidor de aplicações pode definir outros arquivos necessários para a sua implantação
  - Geralmente consistem de um ou mais arquivos XML que devem ser embutidos nos EJB-JARs, WARs ou EARs
  - Nos servidores comerciais (e no J2EE Reference Implementation) os arquivos podem ser gerados e automaticamente incluídos nos JARs
- Os arquivos servem para configurar os componentes para usarem serviços proprietários ou para redefinir valores default (como nomes de contextos JNDI)
- No JBoss, o arquivo jboss.xml é usado para este fim
  - Outros dois arquivos: jboss-web.xml e jbosscmp-jdbc.xml configuram recursos adicionais (ambos são opcionais)
  - O arquivo jboss.xml deve ser incluido no META-INF do EJB-JAR.

Como implantar (deploy) uma aplicação

- Este é um processo dependente do servidor
  - Basicamente, o servidor registra o EJB-JAR, WAR ou EAR, que contém todas as informações necessárias para a implantação do componente guardadas nos seus deployment descriptors
  - Cada fabricante tem uma maneira diferente de fazer esse registro
- No ambiente de desenvolvimento do J2EE (Reference Implementation)
  - Use o deployment wizard na ferramenta deploytool
- No JBoss
  - Simplesmente joque o JAR, WAR ou EAR no diretório "deploy"

Estrutura do pacote Sun J2EE SDK

bin	Executáveis
├ j2ee	Servidor Sun J2EE Reference Implementation
├ cloudscape	Servidor de banco de dados Informix Cloudscape
├ runclient	Ferramenta para execução de clientes com autenticação
├ cleanup	Esvazia o cache
├ verifier	Verificador de JARs (diagnostica erros em JARs, EARs)
├ deploytool	Ferramenta gráfica para deployment e montagem
├ j2eeadmin	Ferramenta de administração (datasources, destinos JMS)
├ packager	Empacotador de JARs, EARs, WARs, RARs
├ realmtool	Gerencia usuários, grupos, etc.
└ keytool	Gerencia chaves de criptografia
cloudscape	Diretório raiz do servidor de banco de dados Cloudscape
doc	Documentação do J2EE SDK
└ api	Documentação da API
config	Arquivos .properties com configuração para aplicações
lib	JARs usados em aplicações J2EE (contém o j2ee.jar)
public_html	Raiz do servidor Web
repository	Repositório de dados e cache do servidor

Configuração: variáveis de ambiente

- Para todas as aplicações
  - ANT_HOME: deve apontar para o diretório de instalação do ANT (ex: c:\jakarta-ant1.5Beta)
  - JAVA_HOME: deve apontar para o diretório de instalação do JSDK (ex: c:\j2sdk1.4.0)
  - PATH contendo ANT_HOME/bin e JAVA_HOME/bin
- Para o J2EE Reference Implementation
  - J2EE_HOME: diretório de instalação do J2SDK EE
  - PATH contendo J2EE_HOME/bin
- Para compilar, use o CLASSPATH
  - $J2EE_HOME/lib/j2ee.jar
- Para executar aplicações, use os JARs indicados pelo seu servidor de aplicações
  - JBoss: JARs específicos localizados em $JBOSS_HOME/lib/ e client/
  - J2EE: $J2EE_HOME/lib/j2ee.jar e $J2EE_HOME/lib/locale

J2EE Reference Implementation

- *É um servidor completo que implementa a especificação J2EE*
  - *Você geralmente não precisa do J2EE SDK se já tem um servidor de aplicação compatível com a especificação J2EE*
- *Contém*
  - *Ferramentas de montagem e instalação para o servidor J2EE RI*
  - *EJB container, App Client container e Servlet container (Tomcat)*
  - *Servidor de banco de dados (Cloudscape)*
  - *Serviços de transações (JTS), persistência (CMP), autenticação e autorização (JAAS) e outros serviços básicos*
- *Para iniciar/parar o servidor*

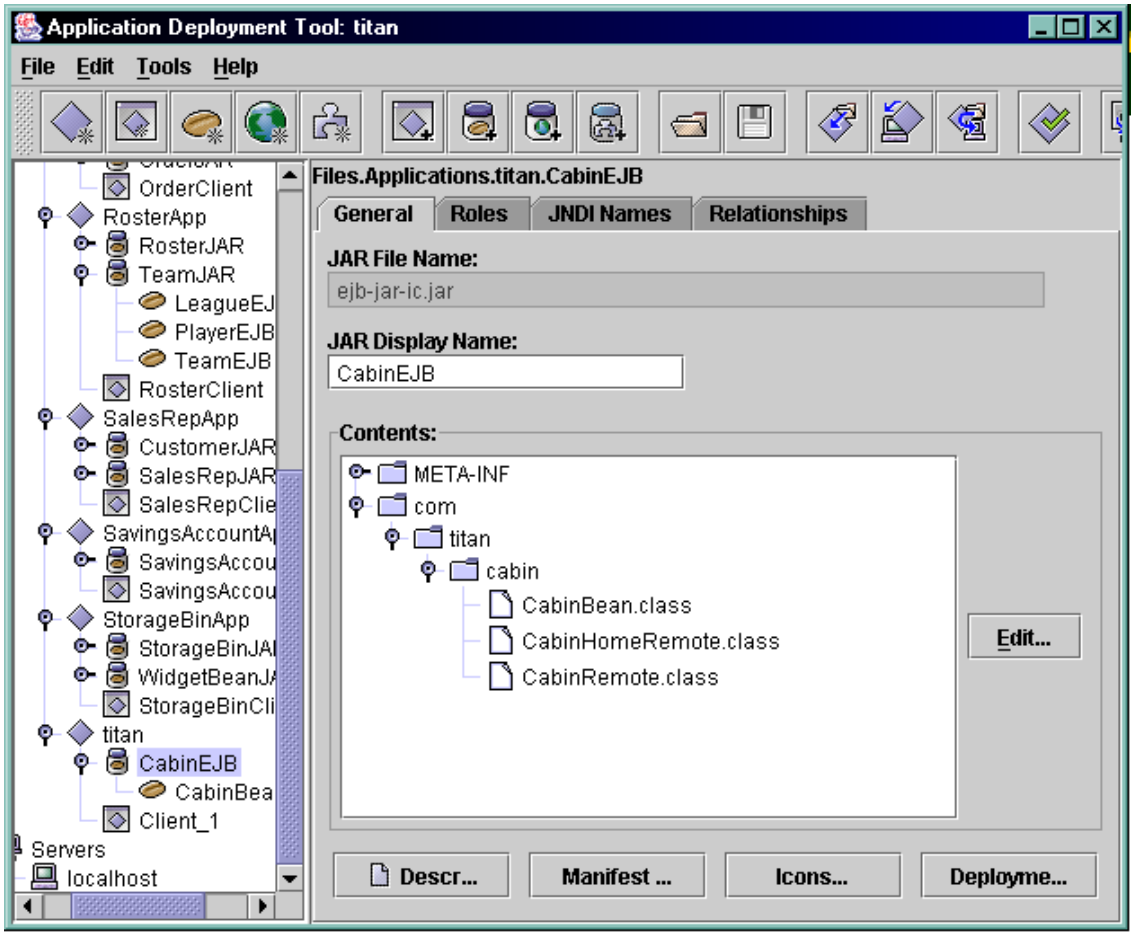
```
> j2ee -verbose      Exibe mensagens durante a execução
> j2ee -stop         Interrompe o servidor
```

- *Web container:*
  - *Porta 8000: servidor Web*
  - *Porta 7000: servidor Web seguro*

J2EE: deployment tool (deploytool)

- *Objetivo: facilitar a criação, montagem, implantação e manutenção de aplicações J2EE no servidor J2EE RI*
  - *Pode também ser usado para montar componentes (WAR, EJB-JAR, EAR, etc.) para outros servidores também (os que não oferecem interface equivalente como o JBoss*)*
- *Para executar:*

```
> deploytool &
```



** neste caso pode ser preciso substituir os arquivos *-ri.xml (do servidor da Sun) gerados por arquivos equivalentes jboss*.xml (não precisa removê-los, basta acrescentar os outros)*

Packager e Administration Tool

- ***packager****: ferramenta de linha de comando para gerar EJB-JARs, WARs, EARs e RARs portáteis (ou não)*

```
> packager -<opção> <parâmetros>
```

  - *Use <opção>* **`-ejbJar`** *para gerar um EJB-JAR*
  - *Use <opção>* **`-webArchive`** *para gerar um WAR*
  - *Use <opção>* **`-enterpriseArchive`** *para gerar um EAR*
- ***j2eeadmin****: ferramenta que adiciona, remove e lista recursos (acessíveis via JNDI) no sistema de nomes servidor J2EE RI*

```
> j2eeadmin -add|-list|-remove<recurso>
```

- ***verifier****: verifica se há erros em um JAR, WAR ou EAR*
- *Veja exemplos de sintaxe do packager, j2eeadmin e outras ferramentas no Apêndice A do Java Tutorial: J2EE SDK Tools*

****Usuários Windows:** Pode haver conflito com o packager.exe e verifier.exe que fazem parte da instalação do Windows. Sugestão: mude os nomes para pack.bat e verify.bat*

ANT: tarefas relacionadas com J2EE

- *Uma forma mais simples (e portável) de criar componentes JAR, WAR, etc. é criar alvos para o Ant em arquivos build.xml:*

```
<ear destfile="app.ear" appxml="application.xml">
   <fileset dir="${build.dir}" includes="*.jar,*.war"/>
</ear>
```

-
```
<jar destfile="${dist}/lib/app.jar">
   <fileset dir="${build}/classes"/>
</jar>
```

-
```
<war destfile="myapp.war" webxml="meta/web.xml">
   <fileset dir="src/jsp/myapp"/>
   <lib dir="jars" /> <classes dir="build/main"/>
   <zipfileset dir="images/gifs" prefix="images"/>
</war>
```

-
```
<ejbjar srcdir="${build}" descriptordir="${xml.dir}">
   <jboss destdir="${deployjars.dir}" />
</ejbjar>
```

- *Mais informações sobre como usar essas tarefas no manual do Ant em $ANT_HOME/docs/manual/index.html*

JBoss 3.0

- Servidor J2EE Open Source
  - Líder de mercado
  - Não certificado oficialmente pela Sun
  - Vencedor do prêmio JavaWorld (concorrendo com BEA, IBM, Sun e outros)
- Onde conseguir
  - **www.jboss.org**
- Instalação e administração
  - Abra o ZIP em algum lugar de sua máquina, mude para o diretório bin da instalação e rode 'run'
  - O JBoss geralmente não precisa de pré-configuração para funcionar

JBoss 3.0 (2)

- Documentação oficial
  - A documentação do JBoss é usada para financiar o projeto Open Source. Na versão atual são três livros de US$10.00 cada um. Para a maior parte dos ambientes, o primeiro livro é suficiente (os outros dois lidam com clustering e implementação de persistência)
- Documentação on-line (não suportada oficialmente)
  - Manual on-line: www.jboss.org/online-manual/HTML/
- Administração e configuração
  - Configuração pode ser feita nos arquivos .xml no diretório conf ou via interface Web (na porta 8082)
  - A interface de administração do JBoss é baseada em JMX (Java Management Extensions): MBeans

Estrutura de diretórios JBoss 3.0.0

bin	Executáveis. Para iniciar o JBoss, rode 'run' neste diretório
server	Contém pastas de servidores
└ default	Servidor default (se você usa clustering tem outros)
├ tmp	Cache do servidor. Contém todas as aplicações instaladas
├ log	Contém os logs de acesso ao container EJB
├ db	Repositório para informações persistentes
├ conf	Arquivos de configuração do servidor (XML/MBean)
├ deploy	Diretório de "hot deployment": Joque seu JAR ou EAR aqui!
└ lib	JARs usados nas aplicações J2EE que rodam no JBoss
client	JARs usados em aplicações-cliente
lib	JARs usados pelo servidor JBoss
docs	Contém os DTDs e exemplos de conectores para serviços
catalina	Diretório raiz do Jakarta-Tomcat (opcional)

- O diretório catalina só existe nas versões com Tomcat
- A estrutura acima é diferente para os servidores JBoss 2.4.x

Exemplos

- Nesta seção iremos demonstrar a construção, montagem e implementação de aplicações J2EE
- Três exemplos
  - Componente EJB: Session bean
  - Componente Web: Página JSP
  - Componente J2EE: Enterprise Archive (EAR)
- O código-fonte de todos os exemplos está em cap01/
- O roteiro que explica os detalhes do código e da implementação está nos livros-texto
  - Veja cópias em PDF e ZIP no subdiretório docs/ebooks e docs/tutorials do CD

Exemplo 1: Componente EJB

- Para este exemplo utilizaremos o roteiro no capítulo 3 do livro-texto "Mastering EJB 2" (consulte) para criar, empacotar e implantar um Session Bean no JBoss
- Os arquivos estão em
  - cap01/mejb2/
- Para montar a aplicação usamos o Ant*

```
> ant buildjboss
```

que monta o ejb-jar. Depois é só copiar para o diretório de deployment do JBoss. O ant faz isto também

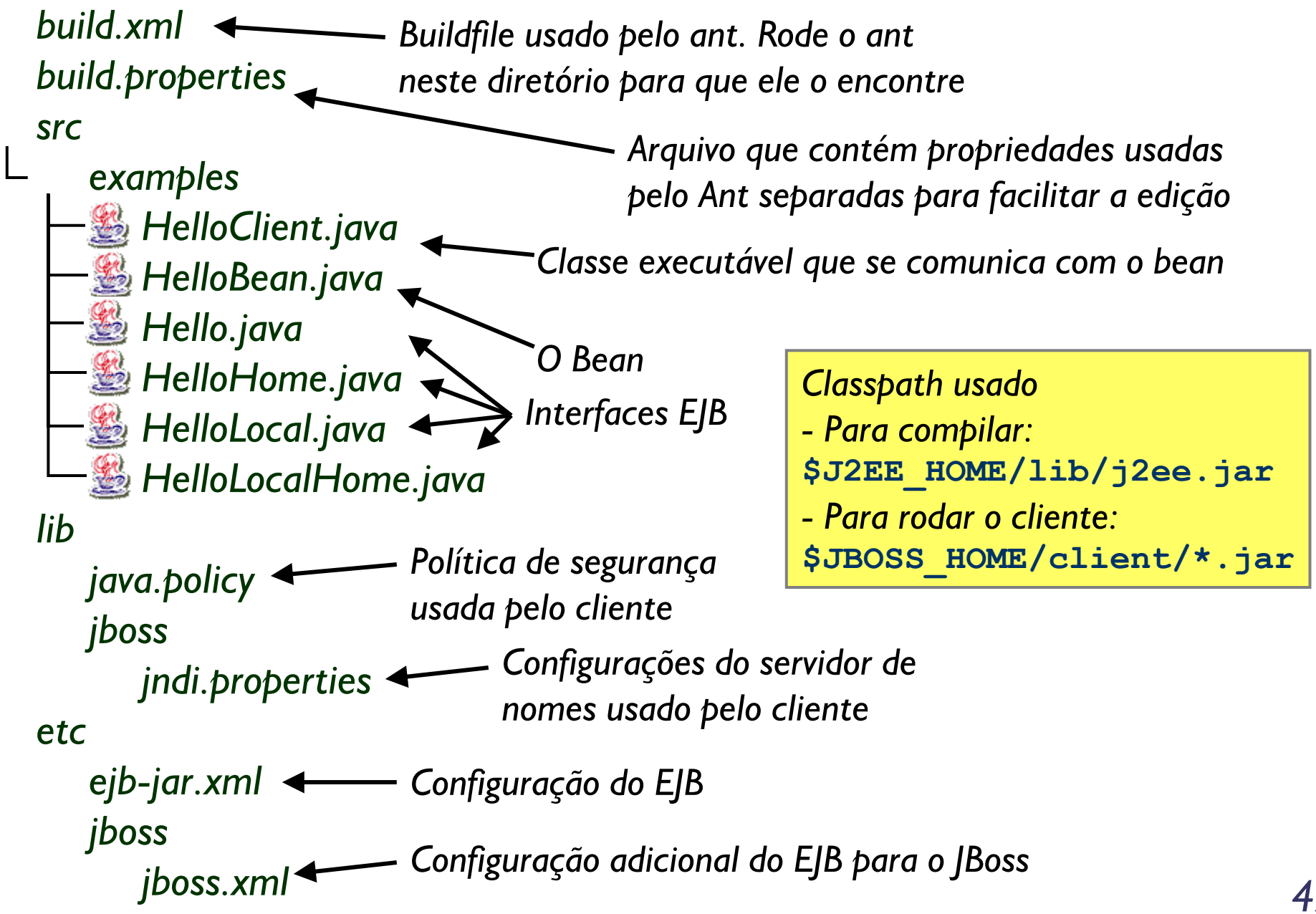
```
> ant jbossdeploy
```

- Para rodar a aplicação cliente (e ver se o EJB funciona) use, mais uma vez, o ant:

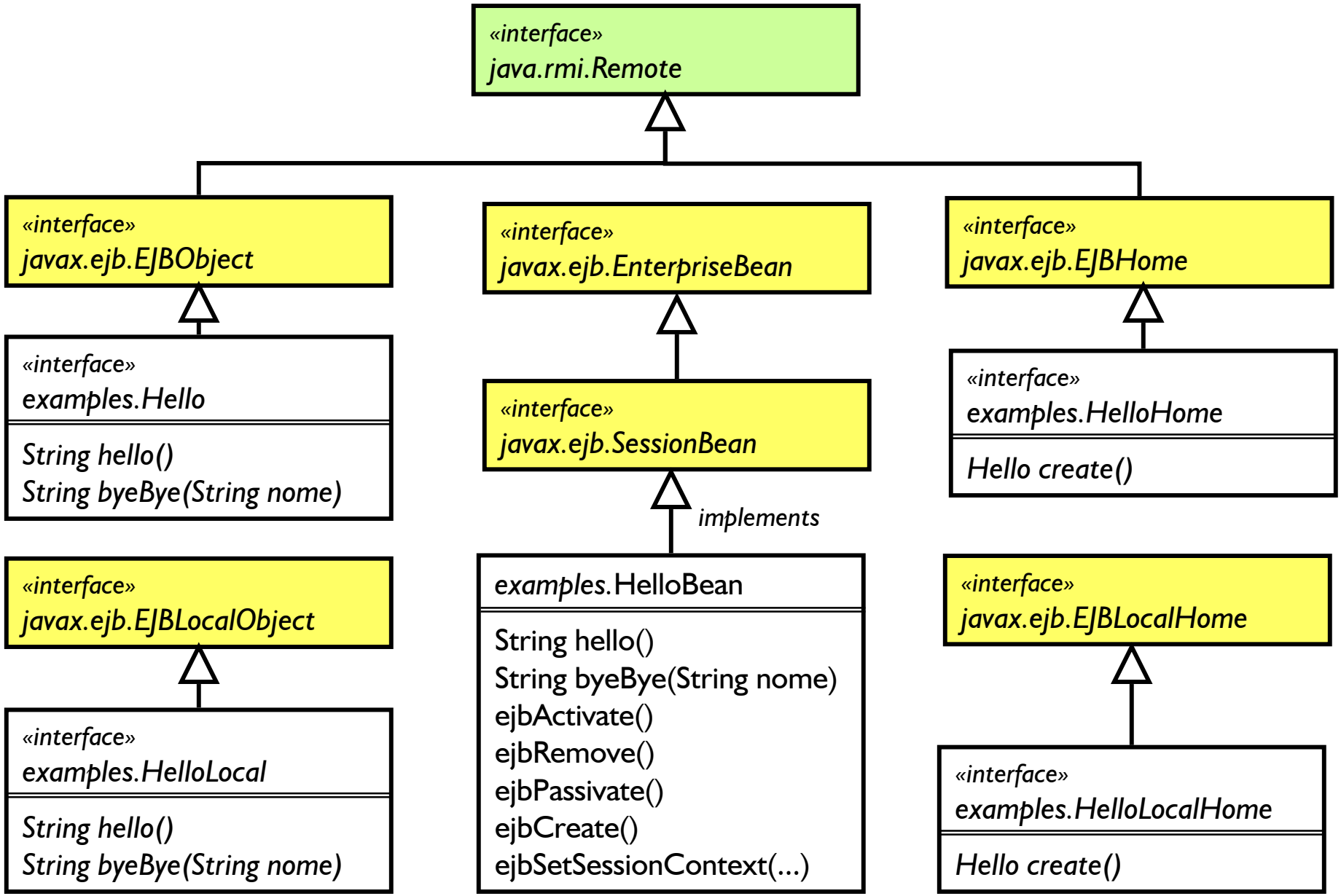
```
> ant runjbossclient
```

* Veja que é preciso configurar o arquivo build.properties com informações do seu sistema

Estrutura da aplicação



Classes e interfaces



Deployment descriptors

```
<!DOCTYPE ejb-jar PUBLIC
"-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
 <enterprise-beans>
  <session>
   <ejb-name>Hello</ejb-name>
   <home>examples.HelloHome</home>
   <remote>examples.Hello</remote>
   <local-home>examples.HelloLocalHome</local-home>
   <local>examples.HelloLocal</local>
   <ejb-class>examples.HelloBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
  </session>
 </enterprise-beans>
</ejb-jar>
```

ejb-jar.xml

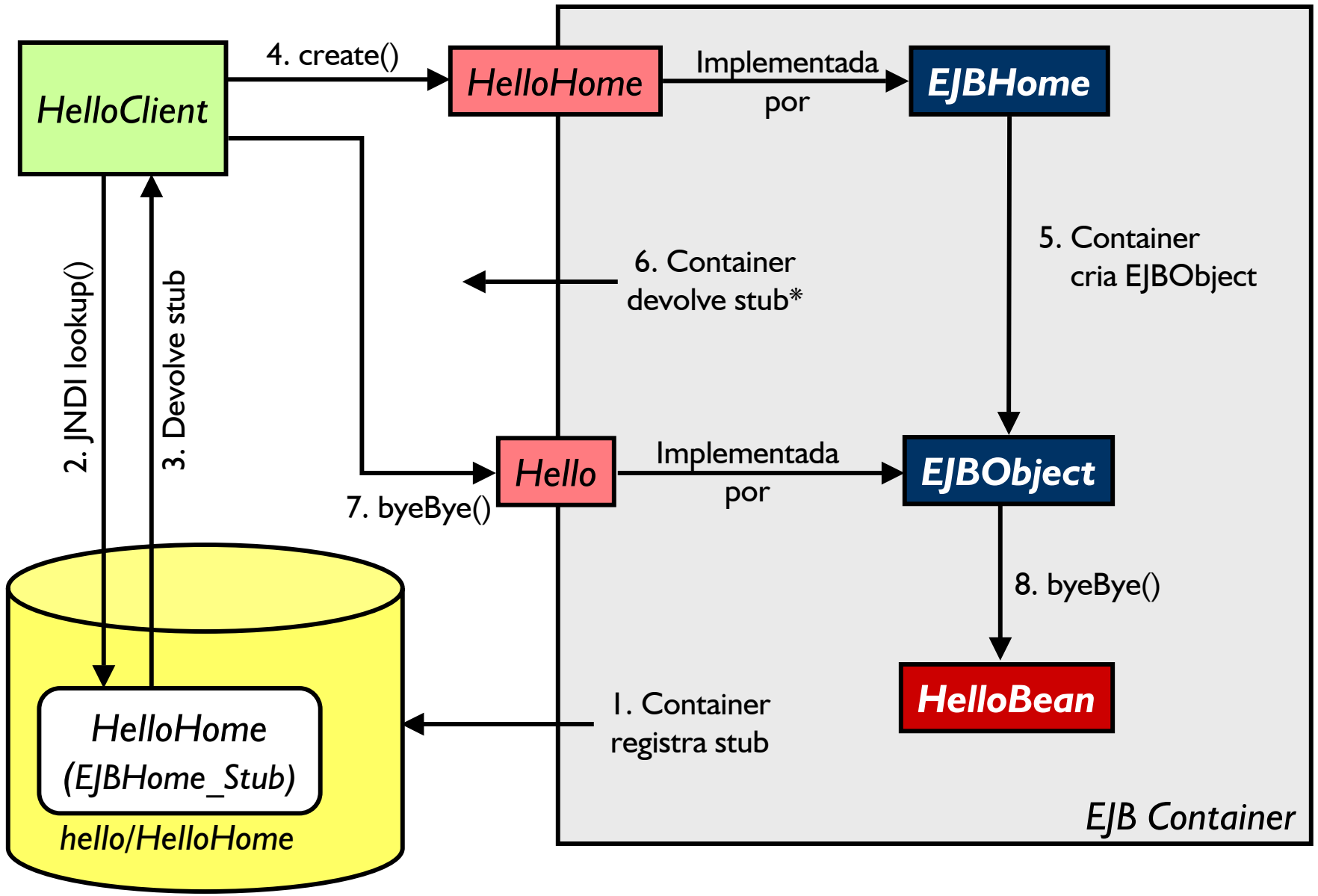
```
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>Hello</ejb-name>
      <jndi-name>hello/HelloHome</jndi-name>
    </session>
  </enterprise-beans>
</jboss>
```

jboss.xml

Configuração do cliente

- O JBoss não oferece um container para a execução do cliente (como faz o servidor da Sun)
- É preciso que a aplicação cliente
  - saiba onde está o servidor de nomes
  - implemente autenticação se necessário
- Se o arquivo jndi.properties estiver no classpath ele será usado pelo cliente. O arquivo contém
  - URL contendo nome do servidor onde roda o serviço de nomes
  - Driver JNDI (nome da classe a ser usada) e pacotes adicionais (é preciso que o JAR onde essas classes estão esteja no CLASSPATH)
- O CLASSPATH do cliente deve enxergar
  - As interfaces Remote e Home do bean
  - JARs que contém drivers de serviços usados (JNDI, JAAS, etc.)
  - Possíveis arquivos de configuração (ex: domínio JAAS)

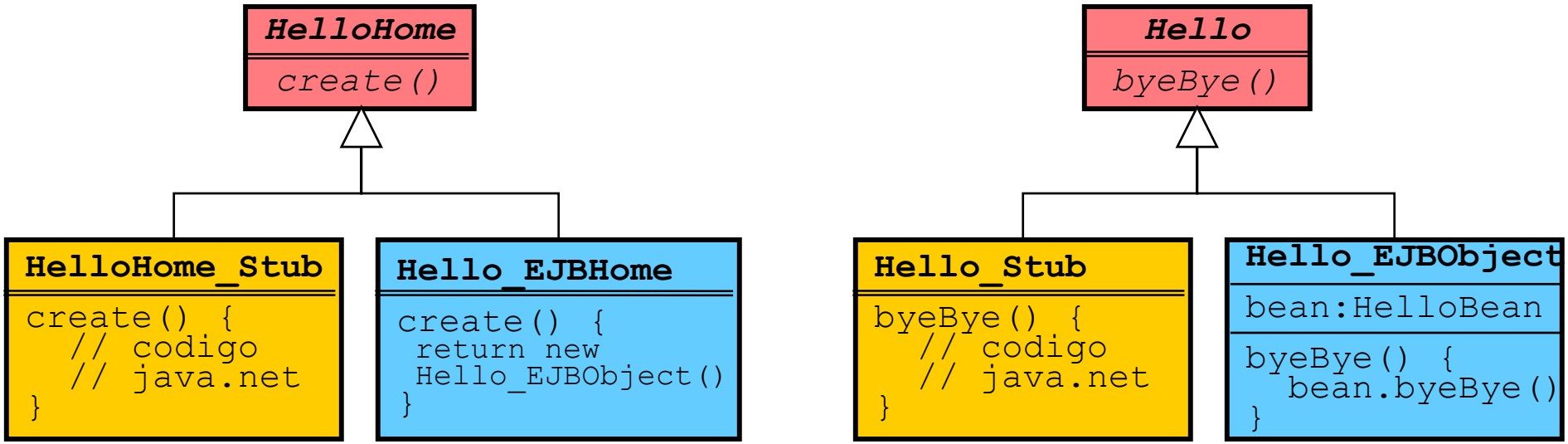
Funcionamento



* *detalhes como comunicação entre stubs e skeletons foram omitidos*

Detalhes

- *Durante o deployment*
  - *Objetos EJBHome e EJBObject são gerados*
  - *Stubs são gerados. Stub da interface Home é mapeado a nome no servidor JNDI*
- *Hierarquias dos objetos gerados pelo container**



- *EJBObject delega requisições para Bean*

** Apenas para efeito de ilustração. Implementação exata depende, na verdade, do fabricante*

Exemplo 2: Componente Web

- Neste exemplo criaremos um componente JSP muito simples e empacotaremos em um arquivo WAR
- 1. Criamos primeiro a seguinte estrutura de diretórios

```
web
└── WEB-INF
```

➡ Use o DTD: web-app_2_3.dtd

- 2. No diretório WEB-INF colocamos o seguinte arquivo web.xml

```
<!DOCTYPE ...> <web-app></web-app>
```
web.xml

- 3. Na pasta web colocamos o seguinte arquivo index.jsp

```
<% String msg = "World!";
   String parameter = null;
   if ( (parameter = request.getParameter("nome")) != null) {
      msg = parameter;
   }
%>
<h1>Hello, <%=msg %></h1>
```
index.jsp

- 4. Jogamos todo o conteúdo de web em um JAR (com extensão WAR)

```
> jar cf hello.war -C web .
```

- 5. Copiamos o arquivo WAR para o diretório deploy do Tomcat ou JBoss
- 6. Abrimos o browser apontando na URL

```
http://localhost:8080/hello/index.jsp?nome=Seu+Nome
```

Exemplo 3: Componente Enterprise

- Combinando os dois exemplos anteriores, montaremos um componente EAR fazendo com que o JSP chame o Session Bean criado no primeiro exemplo.
- Este exemplo é o mesmo mostrado na primeira parte do J2EE Tutorial* (Getting Started) só que vamos implantar o componente no JBoss
- Os arquivos estão em
  - cap01/sun/
- Mais uma vez, automatizamos todo o processo no Ant

```
> ant jboss.deploy
```

- compila tudo, monta o EJB-JAR, o WAR, o EAR e joga no diretório deploy. Para rodar o cliente

```
> ant run.jboss.client
```

* consulte o tutorial para explicações sobre o código-fonte

Conclusão

- *Apesar de serem simples os exemplos vistos neste módulo, eles ilustram todo o processo de desenvolvimento J2EE*
  - *Codificação das interfaces, enterprise bean e componentes Web*
  - *Empacotamento em EJB-JARs, WARs e EARs*
  - *Configuração dos componentes através de deployment descriptors*
  - *Implantação (deployment) em um servidor de aplicações*
- *Já temos, portanto, bons fundamentos teóricos e alguma experiência prática para começar a desenvolver e montar aplicações J2EE*
- *Nos próximos três módulos veremos tecnologias que dão suporte aos componentes J2EE:*
  - *JNDI, que oferece acesso ao serviço de nomes*
  - *RMI-IIOP, que permite a comunicação entre Enterprise Beans, e*
  - *JMS, que permite a comunicação assíncrona através de Enterprise Beans especiais chamados de Message Driven Beans*

Exercícios

Use os arquivos em cap01/exercicio/

- 1. a) Altere o session bean do primeiro exemplo e acrescente novos métodos
  - Um método que retorne a data e a hora
  - Um método que receba dois números e retorne a soma
- 1. b) Empacote o bean e faça o deployment no JBoss
- 2. Escreva uma aplicação cliente (linha de comando) que receba os valores necessários, conecte-se ao servidor e imprima o resultado na tela
- 3. Escreva uma aplicação cliente Web (JSP, empacotado em um WAR) que interaja com o bean e imprima os resultados
  - Use os web.xml, application.xml e ejb-jar.xml fornecidos
  - Use os arquivos semi-prontos (siga as instruções nos comentários)

Empacote tudo em um EAR e faça o deployment no JBoss

Fontes

[1] Bill Shannon. *J2EE Specification*. Sun Microsystems. http://java.sun.com/j2ee/

[2] Richard Monson-Haefel. *Enterprise JavaBeans, 3rd. Edition*. O'Reilly, 2001. *Uma das mais importantes referências sobre EJB*

[3] Sun Microsystems. *Simplified Guide to the J2EE*. http://java.sun.com/j2ee/. *White paper com introdução a J2EE.*

[4] Ed Roman et al. *Mastering EJB 2, Chaps. 1 to 3*
http://www.theserverside.com/books/masteringEJB/index.jsp
Contém ótima introdução a middleware, motivação e fundamentos de EJB

[5] Rossana Lee. *The J2EE Tutorial*, Sun Microsystems.
http://java.sun.com/j2ee/tutorial/. *Roteiro principal dos exemplos e exercícios.*

[6] Kevin Boone, et al. *JBoss User's Manual: Chapter 1: First Steps.*
http://www.jboss.org/online-manual/HTML/ch01.html. *Passo-a-passo para montar e instalar um EJB no JBoss*

[7] Bruce Eckel. *Thinking in Java 2*. http://www.bruceeckel.com. *Capítulo final dá uma visão geral da tecnologia J2EE (versão 1.2)*

[8] Duane Fields e Mark Kolb. *Web Development with JavaServer Pages*, Manning, 2000. *Referência sobre JSP.*

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)

Introdução a J2EE, 2000, 2001, 2002, 2003
Atualizado em Abril de 2003

Serviços de Localização

Helder da Rocha
www.argonavis.com.br

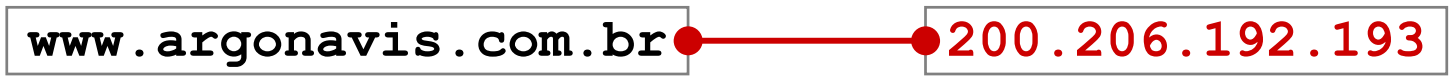
Objetivos

- *Este módulo descreve os serviços básicos para localização de recursos em J2EE*
  - *Localização de componentes (servlets, EJBs)*
  - *Localização de fontes de dados (bancos, filas, pilhas)*
  - *Objetos compartilhados*
- *Recursos são localizados através da interface JNDI - Java Naming and Directory Interface*
  - *Este módulo descreve o uso básico de JNDI*
  - *Para maiores detalhes e sobre o uso de JNDI independente de um servidor de aplicações, consulte os slides do minicurso J523 - JNDI*

Serviço de nomes

- *A principal função de um serviço de nomes é permitir a associação de um nome (ou uma outra representação alternativa mais simples) a recursos computacionais como*
  - *endereços de memória, de rede, de serviços*
  - *objetos e referências*
  - *códigos em geral*
- *Suas duas funções básicas são*
  - *Associar (mapear) um nome a um recurso*
  - *Localizar um recurso a partir de seu nome*
- *Exemplos*
  - *Sistema de arquivos: liga caminho a bloco(s) de memória:*

  - *Sistema DNS: liga nome de domínio a endereço IP:*



Contextos e Sistemas de Nomes

- Um contexto é um conjunto de ligações nome-objeto
  - Em outras palavras, é um objeto que tem zero ou mais ligações
- Se o objeto (referência) contido no contexto for também um contexto ele é um subcontexto
  - O escopo do subcontexto é limitado pelo seu contexto pai
- Exemplos de contextos e subcontextos:
  - /usr/bin/java/

    usr é o contexto; bin é subcontexto de usr, ...
  - www.abc.com.br

    br é o contexto, com é subcontexto de br, ...
- Um sistema de nomes é um conjunto interligado de contextos que respeitam a mesma convenção e possuem um conjunto comum de operações

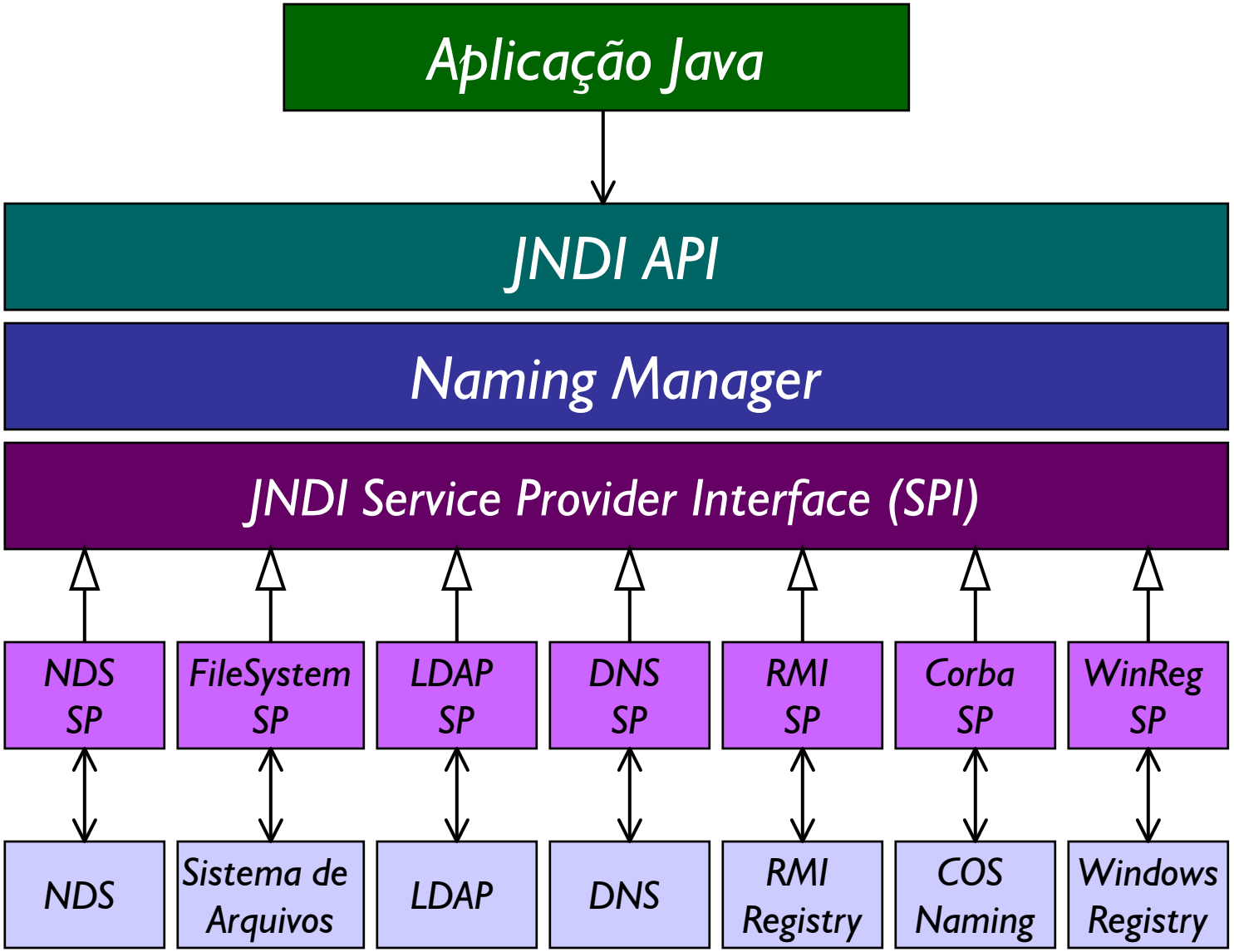
Diretórios e Serviços de Diretório

- *Diretório: conjunto interligado de objetos*
  - *Organização não precisa ser hierárquica (contextual)*
  - *Para cada item há um nome unívoco (chave)*
  - *Cada item possui um ou mais atributos*
- *Um serviço de diretório é oferece operações para criar, remover, modificar e principalmente pesquisar atributos associados a objetos em um diretório*
- *Um atributo possui*
  - *Um identificador*
  - *Conjunto de valores*
  - *Um tipo: restringe os dados que um atributo pode receber*
- *Sistemas de nomes são frequentemente estendidos com serviços de diretório*

JNDI

- ***Java Naming and Directory Interface*** *é uma ponte sobre os diversos serviços de nomes e diretórios diferentes*
- *Vantagens*
  - *Só é preciso aprender uma única API para acessar vários tipos de informação de serviços de diretório*
  - *Isola a aplicação dos detalhes específicos do protocolo*
  - *Pode ser usada para ler objetos Java (serializados) que estejam armazenados em um diretório*
  - *Pode combinar diferentes tipos de diretório (federação) e tratá-los como um diretório único*
- *Componentes*
  - ***API*** *- Application Programming Interface*
  - ***SPI*** *- Service Provider Interface que permite que novos serviços sejam plugados transparentemente*

Arquitetura JNDI



Fonte: JNDI Tutorial

JNDI API

- A API JNDI está incluída no J2SDK 1.3 ou posterior nos pacotes e subpacotes descendentes de javax.naming.
- Para usar JNDI é preciso ter
  - As classes e interfaces do JNDI (pacotes javax.naming.*)
  - Pelo menos um provedor de serviços JNDI (driver)
- O Java 2 SDK inclui provedores de serviço (SPs) para
  - LDAP - Lightweight Directory Access Protocol
  - CORBA - Common ORB Architecture e COS name service
  - Java RMI Registry
- Outros provedores de serviços (para sistema de arquivos, para DNS, para JDBC, para Windows Registry, etc.) podem ser encontrados a partir do site

`http://java.sun.com/products/jndi/serviceproviders.html`

Principais classes

- A API JNDI consiste de cinco pacotes
- O principal pacote é **javax.naming** que contém as principais classes e interfaces
  - Context: interface onde se pode recuperar, ligar, desligar e renomear objetos, e criar e destruir contextos
  - InitialContext: ponto de partida (raiz) para todas as operações
  - Name: abstração de um nome. Contém geralmente um String de texto que corresponde ao nome do objeto ou contexto
  - NameClassPair: contém nome do objeto e de sua classe
  - Binding: contém nome do objeto ligado, nome da classe do objeto e o próprio objeto
  - Reference: abstração de uma referência para um objeto
  - NamingEnumeration: um tipo de java.util.Enumeration usado para colecionar componentes de um contexto
  - NamingException: principal exceção do JNDI

Contexto inicial

- *Precisa ser obtido antes de qualquer operação. Passos:*
- *1: selecionar o provedor de serviços*

```
Properties env = new Properties();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "classe.do.ProvedorDeServicos");
```

- *2: configurar o acesso ao serviço*

```
env.put(Context.PROVIDER_URL, "ldap://xyz.com:389");
env.put(Context.OUTRA_PROPRIEDADE, "valor"); (...)
```

- *3: criar um objeto para representar o contexto*

```
Context ctx = new InitialContext(env);
```

- *A configuração (1, 2) pode ser feita via propriedades do sistema*
  - *passadas em linha de comando via agrumento* `-Dprop=valor`
  - *carregados via arquivos de propriedades*
- *Principais propriedades*

`java.naming.factory.initial:`	`Context.INITIAL_CONTEXT_FACTORY`
`java.naming.provider.url:`	`Context.PROVIDER_URL`

Recuperação de objetos (lookup)

- *Para obter a referência para um objeto de um contexto usa-se o método **lookup()***
  - *Para usar o objeto retornado é preciso conhecer o seu tipo e fazer o cast (ou narrow, se objeto remoto) para promover a referência*
  - *Se o objeto for um contexto, lookup() age como um método para mudar de contexto (como o chdir, em Unix)*
- *Exemplo*
  - *O método **lookup()** usando com o provedor de serviço fscontext retorna um java.io.File pelo nome de arquivo*

Serviço de nomes para sistema de arquivos

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.fscontext.RefFSContextFactory");

env.put(Context.PROVIDER_URL,
        "file:/cap02/lab/filesys");

Context ctx = new InitialContext(env);

File f = (File)ctx.lookup("report.txt");
```

Diretório raiz do serviço

Arquivo localizado na raiz do serviço

Listagem do conteúdo de contextos

- *Em vez de obter um objeto de cada vez via lookup(), pode-se obter uma lista deles*
  - *Método **list()** retorna uma lista (NamingEnumeration) de pares nome / nome_da_classe (objetos do tipo NameClassPair)*

```
NamingEnumeration lista = ctx.list("awt");
while (lista.hasMore()) {
   NameClassPair nc = (NameClassPair)lista.next();
   System.out.println(nc);
}
```

Trecho de List.java

  - *Método **listBindings()** retorna uma lista de ligações nome / objeto (Binding)*

```
NamingEnumeration lista = ctx.listBindings("awt");
while (lista.hasMore()) {
   Binding bd = (Binding)lista.next();
   System.out.println(bd.getName() + ": " + bd.getObject());
}
```

Trecho de ListBindings.java

Modificação de ligações, objetos e contextos

- *Adicionando ligações*

```
Fruit fruit = new Fruit("orange");
ctx.bind("favorite", fruit);
```
Bind.java

- *Substituindo ligações*

```
Fruit fruit = new Fruit("lemon");
ctx.rebind("favorite", fruit);
```
Rebind.java

- *Removendo ligações*

```
ctx.unbind("favorite");
```
Unbind.java

- *Renomeando objetos*

```
ctx.rename("report.txt", "old_report.txt");
```
Rename.java

- *Criando novos contextos*

```
Context result = ctx.createSubcontext("new");
```
Create.java

- *Destruindo contextos*

```
ctx.destroySubcontext("new");
```
Destroy.java

Rode List ou ListBindings após cada operação para ver os resultados

Provedores de serviços para objetos

- Pode-se usar JNDI para mapear nomes a objetos
  - Objetos localizáveis por nome podem ser abstraídos do contexto ou até linguagem em que são usados
  - Aplicações diferentes podem compartilhar objetos
- Dois drivers JNDI estão disponíveis para acesso a objetos distribuídos no J2SDK
  - SPI CORBA (COS Naming): permite localização de objetos CORBA (serializados em um formato independende de linguagem)

    `com.sun.jndi.cosnaming.CNCtxFactory`
  - SPI RMI: permite a localização de objetos Java serializados (objetos pode ser usados por outras aplicações Java)

    `com.sun.jndi.rmi.registry.RegistryContextFactory`

Objetos RMI

- *Propriedades a definir*

`java.naming.factory.initial` ou `Context.INITIAL_CONTEXT_FACTORY`

`com.sun.jndi.rmi.registry.RegistryContextFactory`

`java.naming.provider.url` ou `Context.PROVIDER_URL`

`rmi://nome_do_host:1099 (endereço e porta do RMI Registry)`

- *Mapeamento*

```
Context ctx = new InitialContext();
Fruit fruit = new Fruit("orange");
ctx.bind("favorite", fruit);
```

- *Localização*

```
Context ctx = new InitialContext();
Fruit fruit = (Fruit) ctx.lookup("favorite");
```

Objetos CORBA (RMI sobre IIOP)

- *Proprieades a definir*

```
java.naming.factory.initial ou Context.INITIAL_CONTEXT_FACTORY
    com.sun.jndi.cosnaming.CNCtxFactory
java.naming.provider.url ou Context.PROVIDER_URL
    iiop://nome_do_host:1900 (endereço e porta do ORB)
```

- *Mapeamento*

```
Context ctx = new InitialContext();
Fruit fruit = new Fruit("orange");
ctx.bind("favorite", fruit);
```

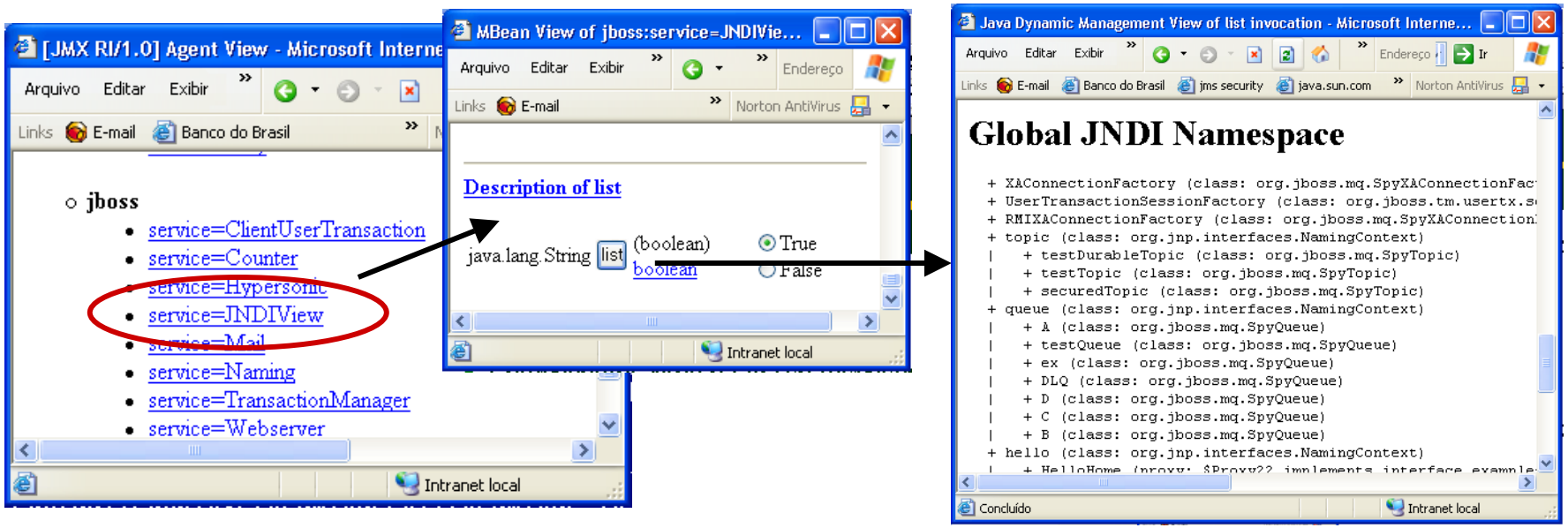
- *Localização*

```
Context ctx = new InitialContext();
Object corbaFruit = ctx.lookup("favorite");
Object javaFruit =
  javax.rmi.PortableRemoteObject.narrow(corbaFruit,
                                        Fruit.class);
Fruit fruit = (Fruit)javaFruit;
```

Converte objeto CORBA em objeto Java

Serviço JNDI no JBoss

- Pode-se usar *bind()* e *lookup()* para armazenar e localizar objetos no servidor JNDI do JBoss
  - Componentes Web e EJB são mapeados (bound) automaticamente, durante a implantação (o servidor usa informações no *web.xml*, *ejb-jar.xml* e outros arquivos de configuração existentes)
- Para verificar os mapeamentos JNDI existentes, acesse o serviço *JNDIView* em *http://localhost:8082*



Fontes de dados

- *No JBoss, pools de conexões de bancos de dados são acessíveis através de objetos DataSource, publicadas no JNDI abaixo no namespace java:/*
  - `java:/CloudscapeDB`
  - `java:/OracleDB`
- *Os nomes são configurados nos arquivos -service.xml do JBoss (localizados em deploy)*
- *Para acessar um banco*

```
Context ctx = new InitialContext();
javax.sql.DataSource ds = (javax.sql.DataSource)
    ctx.lookup("java:/Cloudscape");
java.sql.Connection = ds.getConnection();
```

Enterprise Naming Context

- *Cada EJB e cada aplicação Web tem um namespace próprio para compartilhar objetos usando JNDI*

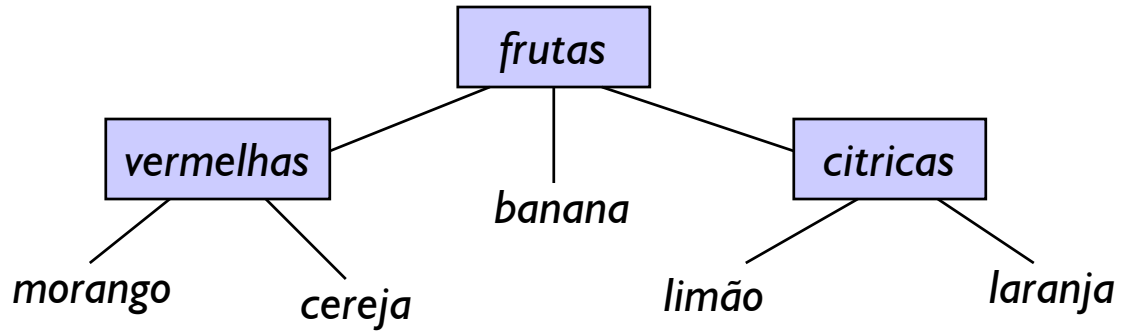
  `java:comp/env`

- *É chamado de Enterprise Naming Context (ENC)*
  - *É mapeado ao JNDI global durante a implantação: permite que componentes de uma mesma aplicação firmem ligações entre si sem depender de recursos externos*
  - *Acesso via java:comp/env não é compartilhado: cada bean ou aplicação Web tem o seu*
- *Forma recomendada de comunicação em J2EE*
  - *Componentes Web e EJBs devem obter referências uns para os outros através de java:comp/env*

```
Objeto obj =
    (Objeto)ctx.lookup("java:comp/env/meuObjeto");
```

Exercícios

- *1. Crie a seguinte hierarquia de contextos e objetos (Fruit) usando as classes fornecidas e o driver JNDI do JBoss e imprima o resultado*



- *2. Localize a estrutura através do JNDIView do JBoss*

Fontes para este capítulo

[1] *Rossana Lee. The JNDI Tutorial, Sun Microsystems, 2002*
http://java.sun.com/products/jndi/tutorial/
A maior parte deste capítulo é baseada nas primeiras duas seções (trilhas) do JNDI Tutorial.

[2] *Ed Roman et al. Mastering EJB 2.0, Wiley, 2001*
http://www.theserverside.com/books/masteringEJB/index.jsp
Apêndice A tem um breve e objetivo tutorial sobre JNDI

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)

Introdução a J2EE, 2000, 2001, 2002, 2003
Atualizado em Abril de 2003

Java 2 Enterprise Edition

Fundamentos de RMI-IIOP

Helder da Rocha
www.argonavis.com.br

Objetivo

- *O objetivo deste módulo é fornecer os pré-requisitos de computação distribuída necessários à eficiente utilização da tecnologia EJB*
  - *Background mínimo sobre Java RMI e Java IDL (implementação de CORBA em Java)*
  - *Desenvolvimento com Java RMI-IIOP (tecnologia usada na comunicação entre EJBs). Consiste do modelo de programação Java RMI com protocolo CORBA (IIOP)*
- *Os "livros-texto" para este módulo são*
  - *Os apêndices **A** e **B** do livro **Mastering EJB 2** de Ed Roman (veja referências no final)*
  - *Os tutoriais da Sun sobre os dois assuntos (nos docs JSDK)*

RMI e RPC

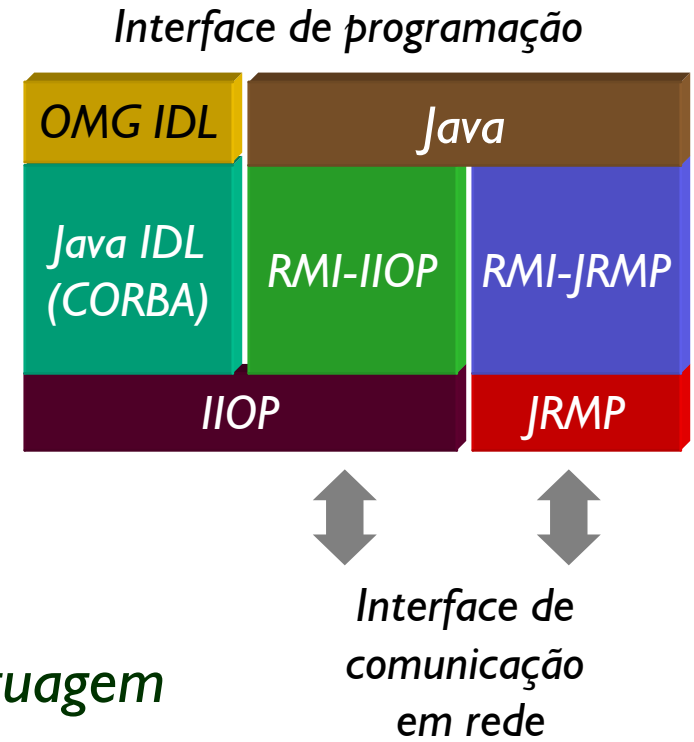
- *RMI e RPC são técnicas usadas para isolar, dos clientes e servidores, os detalhes da comunicação em rede*
  - *Utilizam protocolos padrão, stubs e interfaces*
  - *Lidam com diferentes representação de dados*
- *RPC: Remote Procedure Call*
  - *Chamada procedural de um processo em uma máquina para um processo em outra máquina.*
  - *Permitem que os procedimentos tradicionais permanecam em múltiplas máquinas, porém consigam se comunicar*
- *RMI: Remote Method Invocation*
  - *É RPC em versão orientada a objetos*
  - *Permite possível chamar métodos em objetos remotos*
  - *Beneficia-se de características do paradigma OO: herança, encapsulamento, polimorfismo*

Dificuldades em RPC

- Para dar a impressão de comunicação local, implementações de RPC precisam lidar com várias dificuldades, entre elas
  - Marshalling e unmarshalling (transformação dos dados em um formato independente de máquina)
  - Diferenças na forma de representação de dados entre máquinas
- Implementações RMI tem ainda que decidir como lidar com particularidades do modelo OO:
  - Como implementar e controlar referências dinâmicas remotas (herança e polimorfismo)
  - Como garantir a não duplicação dos dados e a integridade do encapsulamento?
  - Como implementar a coleta de lixo distribuída?
  - Como implementar a passagem de parâmetros que são objetos (passar cópia rasa, cópia completa, referência remota)?
- Padrões diversos. Como garantir a interoperabilidade?

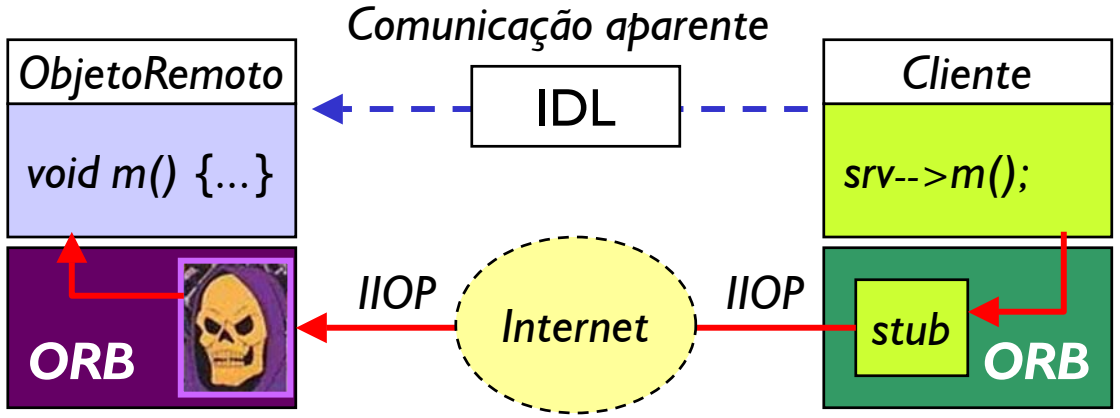
Objetos distribuídos em Java

- *Java RMI sobre JRMP*
  - *Protocolo Java nativo (Java Remote Method Protocol) - Java-to-Java*
  - *Serviço de nomes não-hierárquico e centralizado*
  - *Modelo de programação Java: interfaces*
- *Java IDL: mapeamento OMG IDL-Java*
  - *Protocolo OMG IIOP (Internet Inter-ORB Protocol) independente de plataforma/linguagem*
  - *Serviço de nomes (COS Naming) hierárquico, distribuído e transparente quanto à localização dos objetos*
  - *Modelo de programação CORBA: OMG IDL (language-neutral)*
- *Java RMI sobre IIOP*
  - *Protocolo OMG IIOP, COS Naming, transparência de localidade, etc.*
  - *Modelo de programação Java com geração automática de OMG IDL*



Comunicação CORBA: ORB

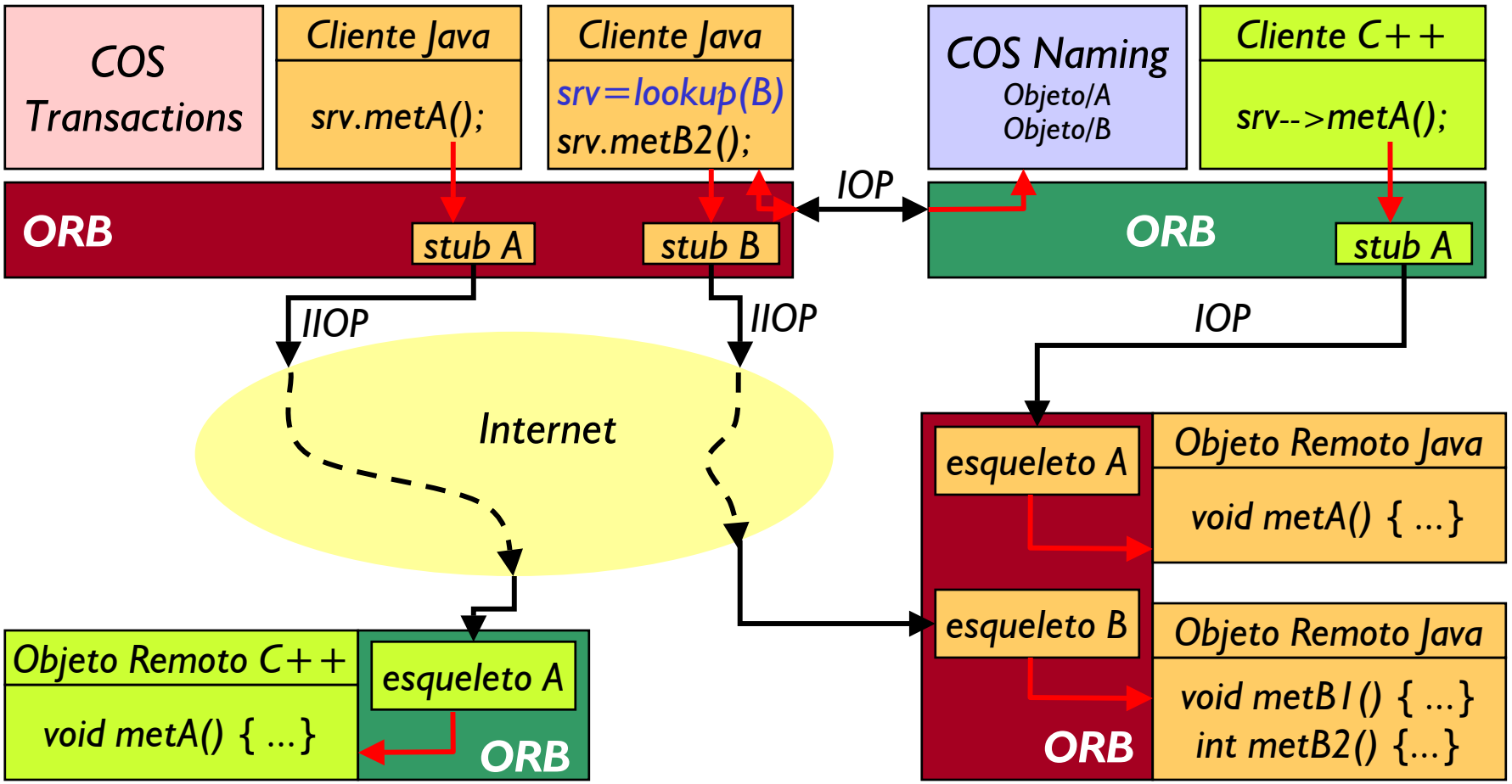
- **IDL**
  - Abstração da interface do objeto remoto
  - Usada para gerar stubs e esqueletos



- **Stub** (lado-cliente)
  - Transforma os parâmetros em formato independente de máquina (marshalling) e envia requisições para o objeto remoto através do ORB passando o nome do método e os dados transformados
- **ORB**: barramento comum
  - ORB do cliente passa dados via IIOP para ORB do servidor
- **Esqueleto** (lado-servidor)
  - Recebe a requisição com o nome do método, decodifica (unmarshals) os parâmetros e os utiliza para chama o método no objeto remoto
  - Transforma (marshals) os dados retornados e devolve-os para o ORB

Comunicação CORBA: IIOP e serviços

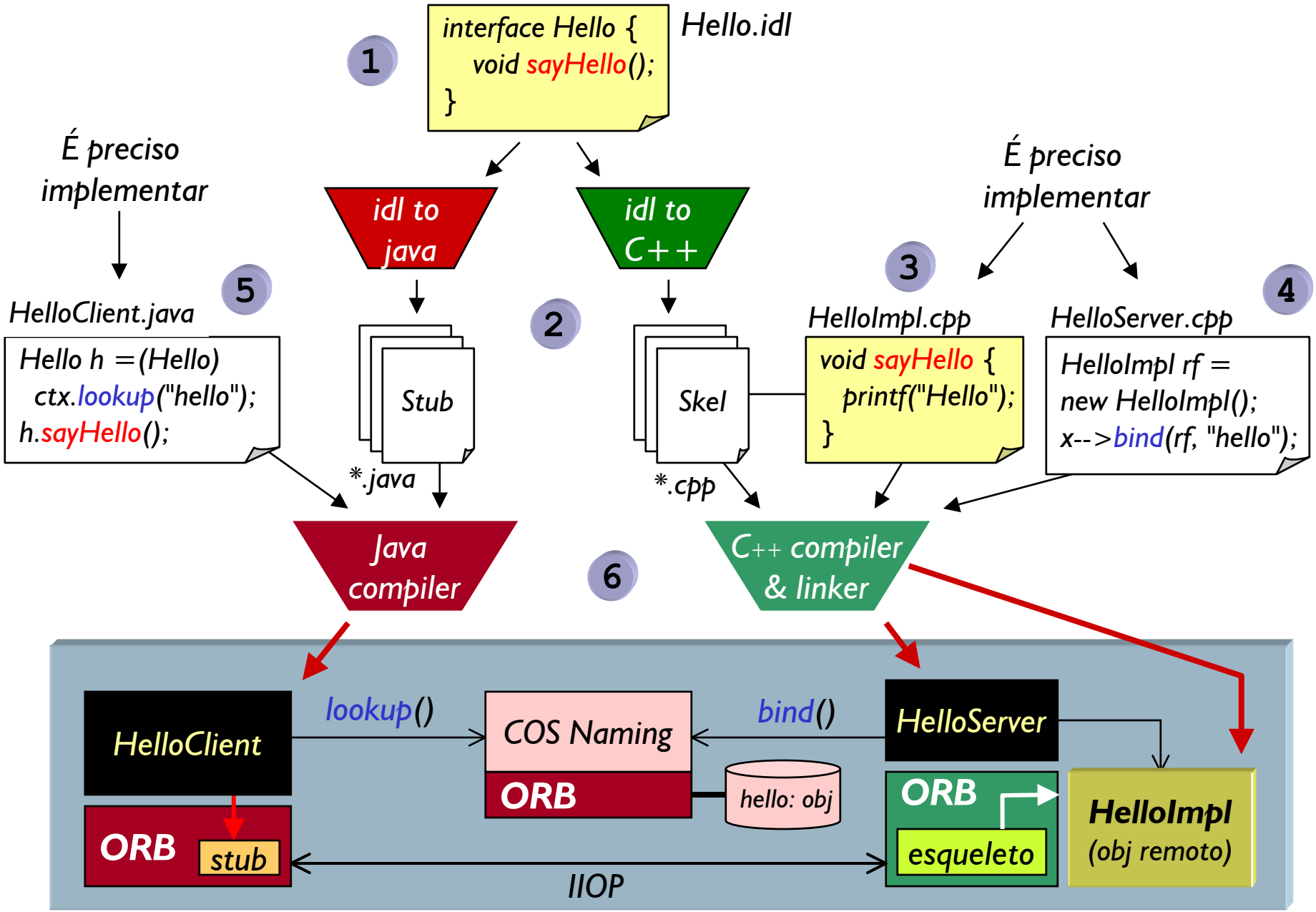
- *Com vários ORBs se comunicando via IIOP, clientes tem acesso a objetos remotos e serviços em qualquer lugar*
  - *ORB encarrega-se de achar o objeto (location transparency)*



Como criar uma aplicação CORBA

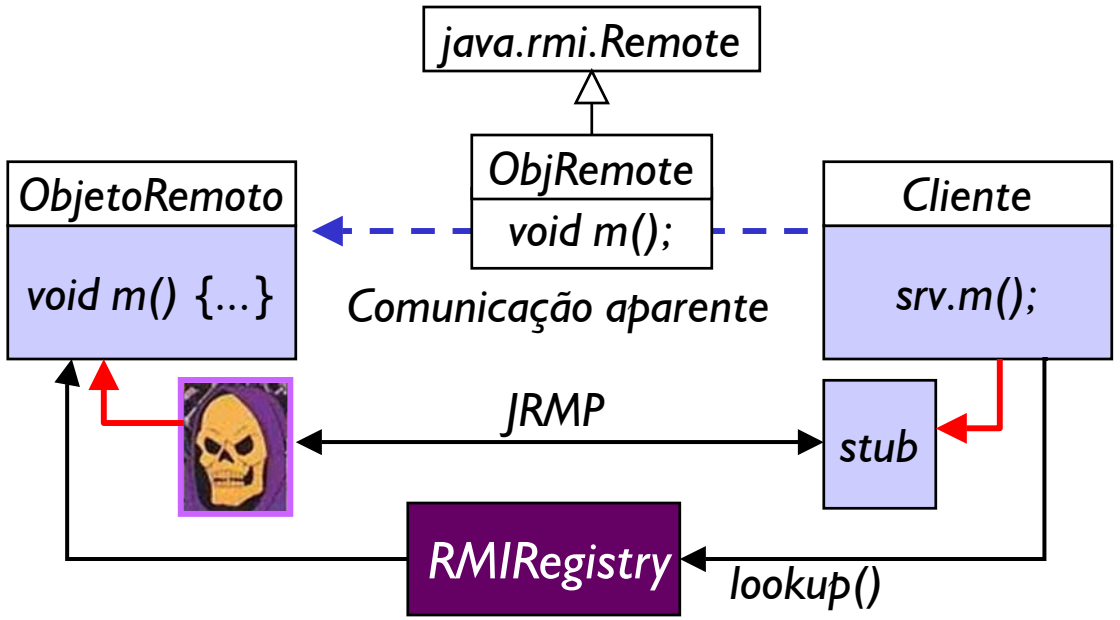
1. Criar uma *interface* para cada objeto remoto em *IDL*
2. *Compilar a IDL* para gerar código de *stubs* e *skeletons*
   - *Usando ferramenta do ORB (em JavaIDL: idlj)*
3. Implementar os *objetos remotos*
   - *Modelo de herança: objeto remoto herda do esqueleto gerado*
   - *Modelo de delegação (mais flexível): classe "Tie" implementa esqueleto e delega chamadas à implementação*
4. Implementar a *aplicação servidora*
   - *Registrar os objetos remotos no sistema de nomes (e usa, opcionalmente, outros serviços)*
5. Implementar o *cliente*
   - *Obter o contexto do serviço de nomes (COS Naming)*
   - *Obtém o objeto remoto através de seu nome*
   - *Converter objeto para tipo de sua interface: xxxHelper.narrow(obj)*
6. *Compilar* e gerar os executáveis

Do IDL a uma aplicação distribuída



Comunicação RMI

- **Interface Remote**
  - Abstração da interface do objeto remoto
  - Usada para gerar stubs e esqueletos
- **Stub** (lado-cliente)
  - Transforma parâmetros (serializados) e envia requisição pela rede (sockets)

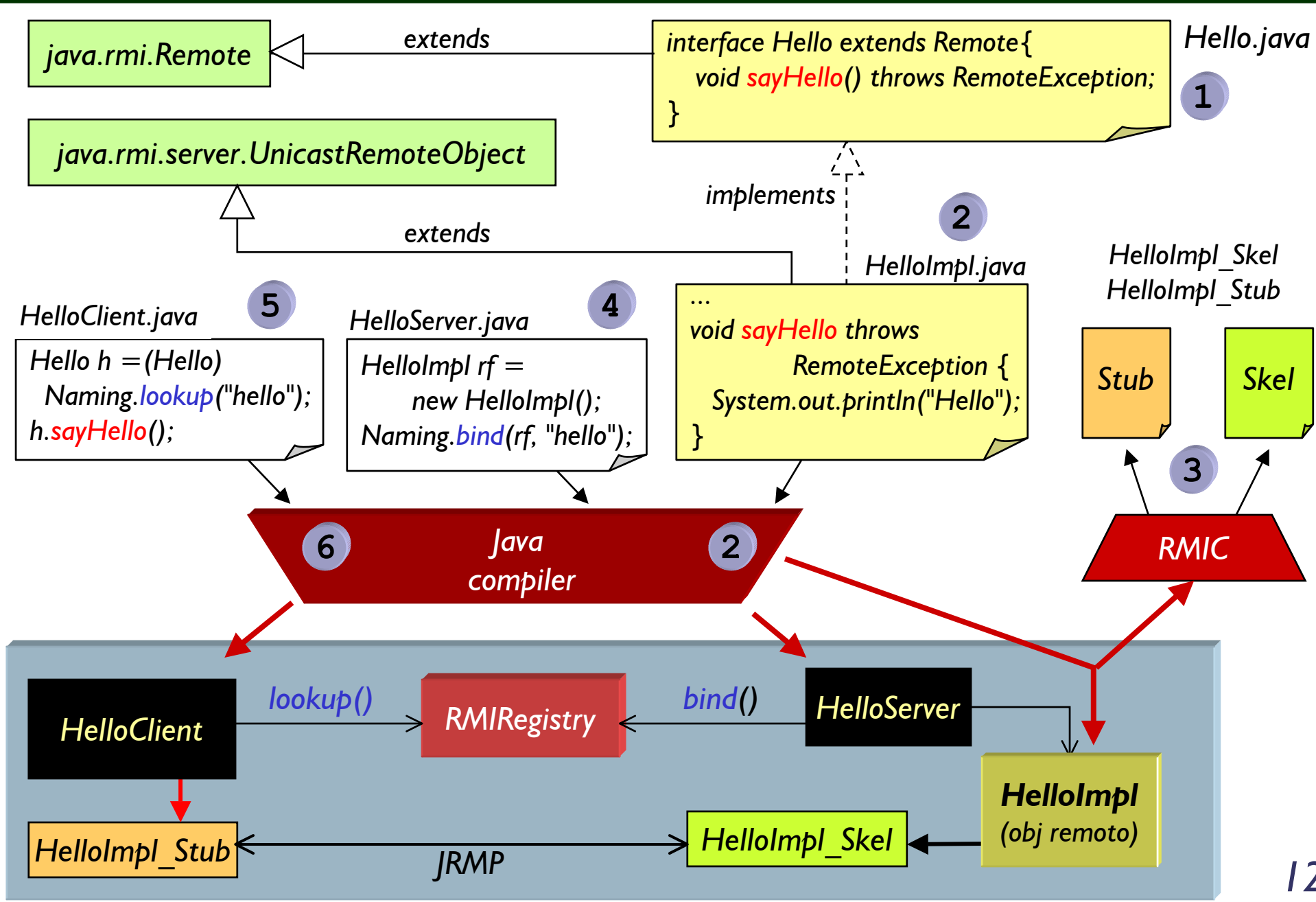


- **Esqueleto** (lado-servidor)
  - Recebe a requisição, desserializa os parâmetros e chama o método no objeto remoto. Depois serializa objetos retornados.
- **RMIRegistry**
  - Servidor registra o objeto usando *java.rmi.Naming.bind()*
  - Cliente recupera o objeto usando *java.rmi.Naming.lookup()*

Como criar uma aplicação RMI

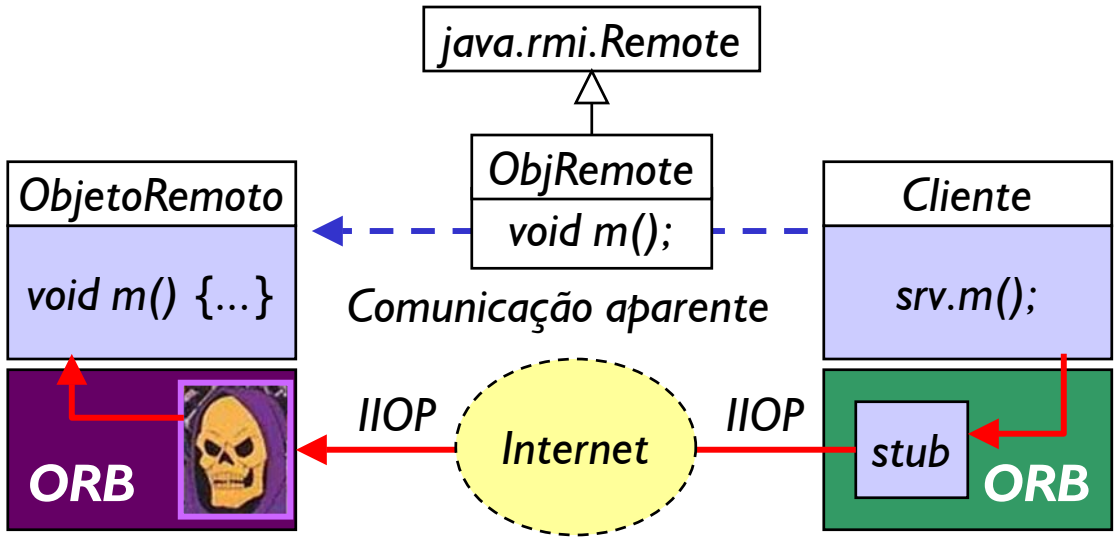
1. Criar subinterface de *java.rmi.Remote* para cada objeto remoto
   - Interface deve declarar todos os métodos visíveis remotamente
   - Todos os métodos devem declarar *java.rmi.RemoteException*
2. Implementar e compilar os *objetos remotos*
   - Criar classes que implementem a interface criada em (1) e estendam *java.rmi.server.UnicastRemoteObject* (ou Activatable)
   - Todos os métodos (inclusive construtor) provocam *RemoteException*
3. Gerar os *stubs* e *skeletons*
   - Rodar a ferramenta *rmic* sobre a classe de cada objeto remoto
4. Implementar a *aplicação servidora*
   - Registrar os objetos remotos no RMIRegistry usando *Naming.bind()*
   - Definir um SecurityManager (obrigatório p/ download de cód. remoto)
5. Implementar o *cliente*
   - Definir um SecurityManager e conectar-se ao *codebase* do servidor
   - Recuperar os objetos remotos usando *(Tipo)Naming.lookup()*
6. *Compilar* servidor e cliente

Construção de aplicação RMI



Comunicação RMI-IIOP

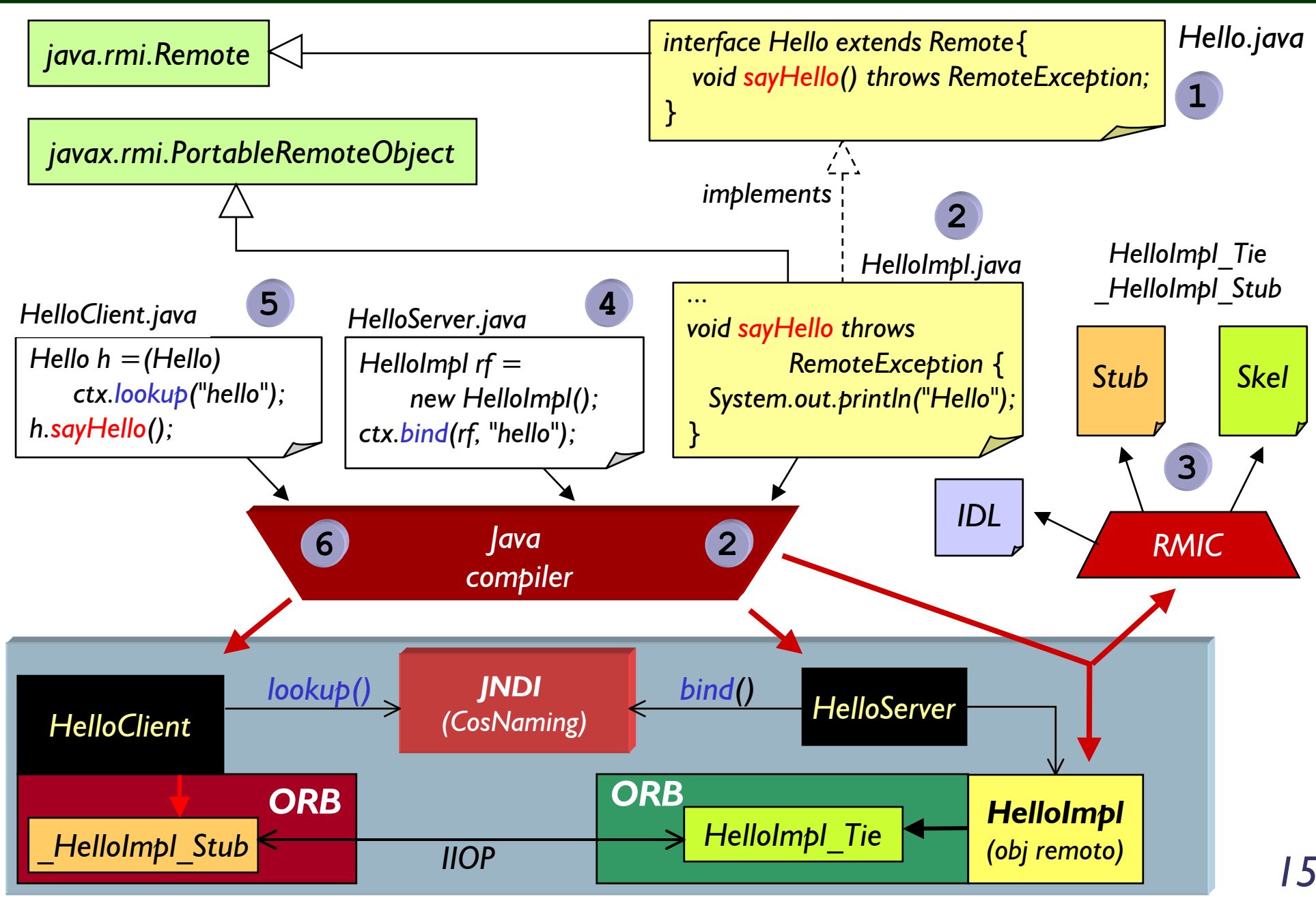
- **Interface Remote**
  - Abstração da interface do objeto remoto
  - Usada para gerar stubs e esqueletos
- **Stub** (lado-cliente)
  - Transforma parâmetros (serializados) em formato independente de máquina e envia requisição pela rede através do ORB
- **ORB**: barramento comum
  - ORB do cliente passa dados via IIOP para ORB do servidor
- **Esqueleto** (lado-servidor)
  - Recebe a requisição do ORB e desserializa os parâmetros
  - Chama o método do objeto remoto
  - Transforma os dados retornados e devolve-os para o ORB



Como criar uma aplicação RMI-IIOP

1. Criar subinterface de *java.rmi.Remote* para cada objeto remoto
   - Interface deve declarar todos os métodos visíveis remotamente
   - Todos os métodos devem declarar *java.rmi.RemoteException*
2. Implementar e compilar os objetos remotos
   - Criar classes que implementem a interface criada em (1) e estendam *javax.rmi.PortableRemoteObject*
   - Todos os métodos (inclusive construtor) provocam *RemoteException*
3. Gerar os *stubs* e *skeletons* (e opcionalmente, os IDLs)
   - Rodar a ferramenta *rmic -iiop* sobre a classe de cada objeto remoto
4. Implementar a aplicação servidora
   - Registrar os objetos remotos no COSNaming usando *JNDI*
   - Definir um SecurityManager
5. Implementar o cliente
   - Definir um SecurityManager e conectar-se ao *codebase* do servidor
   - Recuperar objetos usando *JNDI* e *PortableRemoteObject.narrow()*
6. Compilar

Construção de aplicação RMI-IIOP



Interface Remote

- *Declara os métodos que serão visíveis remotamente*
  - *Todos devem declarar provocar RemoteException*
- *Indêntica à interface usada em RMI sobre JRMP*

```
package hello.rmiop;

import java.rmi.*;

public interface HelloRemote extends Remote {

    public String sayHello()
                  throws RemoteException ;
    public void sayThis(String toSay)
                  throws RemoteException;

}
```

Implementação do objeto remoto

```
package hello.rmiop;

import java.rmi.*;

public class HelloImpl
                    extends javax.rmi.PortableRemoteObject
                    implements HelloRemote {

    private String message = "Hello, World!";
    public HelloImpl() throws RemoteException { }

    public String sayHello()
                    throws RemoteException {
        return message;
    }
    public void sayThis(String toSay)
                    throws RemoteException {
        message = toSay;
    }
}
```

Classe base para todos os objetos remotos RMI-IIOP

Construtor declara que pode provocar RemoteException!

Geração de Stubs e Skeletons

- *Tendo-se a implementação de um objeto remoto, pode-se gerar os stubs e esqueletos*

```
> rmic -iiop hello.rmiop.HelloImpl
```

- *Resultados*
  - *_HelloRemote_Stub.class*
  - *_HelloImpl_Tie.class - este é o esqueleto!*

- *Para gerar, opcionalmente, uma (ou mais) interface IDL compatível use a opção -idl*

```
> rmic -iiop -idl hello.rmiop.HelloImpl
```

- *Resultado*

HelloRemote.idl

```
#include "orb.idl"
module hello {
  module rmiop {
    interface HelloRemote {
       ::CORBA::WStringValue sayHello( );
       void sayThis(in ::CORBA::WStringValue arg0 );
    };
  };
};
```

Tipos definidos em orb.idl (equivalem a wstring)

Servidor e rmi.policy

```
package hello.rmiop;
import java.rmi.*;
import javax.naming.*;

public class HelloServer {
   public static void main (String[] args) {
      if (System.getSecurityManager() == null)
         System.setSecurityManager(new RMISecurityManager());
      try {
         HelloRemote hello = new HelloImpl();
         Context initCtx = new InitialContext(System.getProperties());
         initCtx.rebind("hellormiop", hello);
         System.out.println("Remote object bound to 'hellormiop'.");
      } catch (Exception e) {
         if (e instanceof RuntimeException)
            throw (RuntimeException)e;
         System.out.println("" + e);
      }
   }
}
```

SecurityManager viabiliza download de código

Informações de segurança e serviço de nomes e contexto foram inicial passadas na linha de comando

Associando o objeto com um nome no serviço de nomes

Arquivo de politicas de segurança para uso pelo SecurityManager

rmi.policy

```
grant {
    permission java.net.SocketPermission "*:1024-65535", "connect, accept, resolve";
    permission java.net.SocketPermission "*:80", "connect, accept, resolve";
    permission java.util.PropertyPermission "*", "read, write";
};
```

Cliente

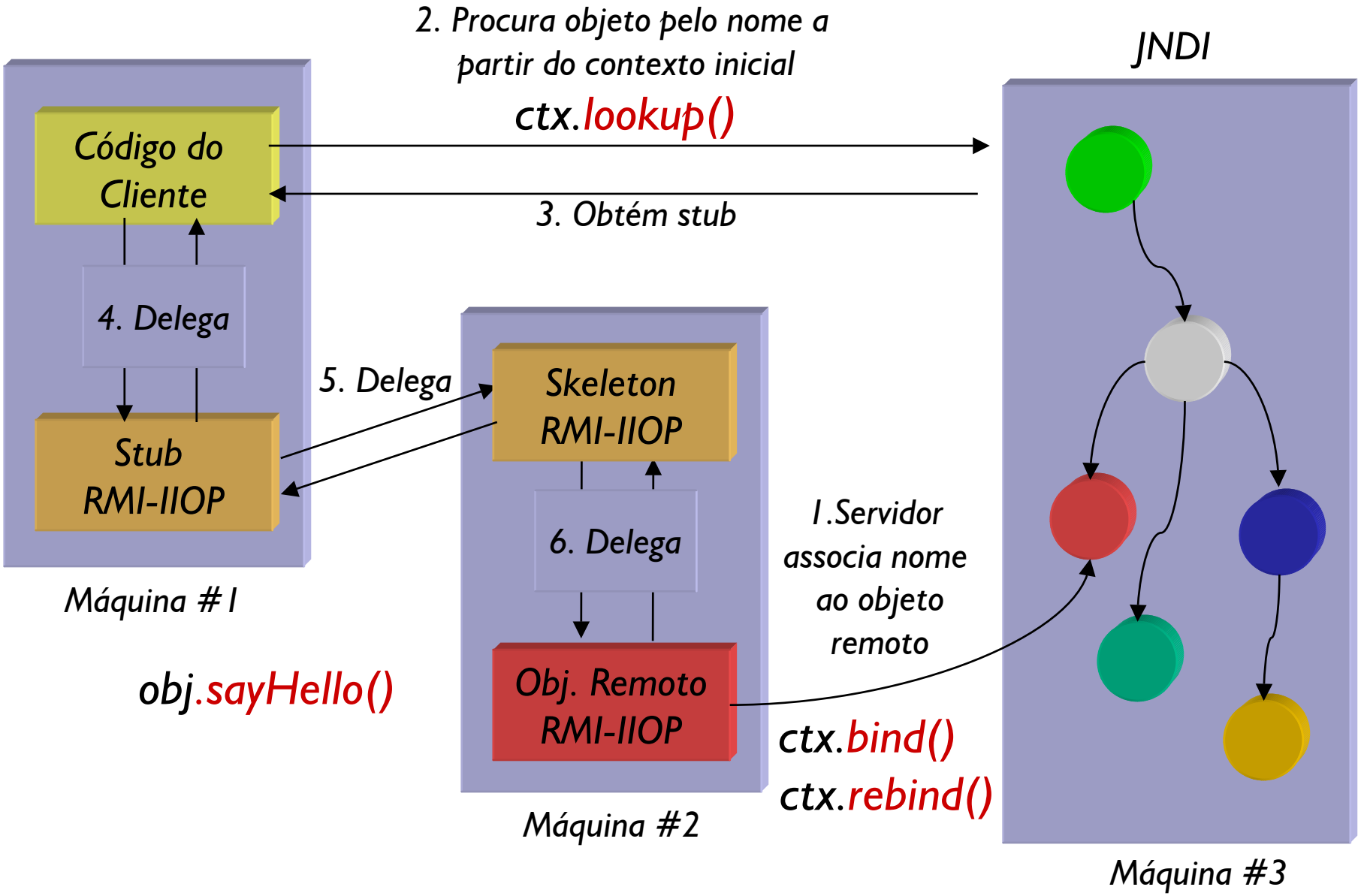
```
package hello.rmiop;

import java.rmi.*;
import javax.naming.*;

public class HelloClient {
  public static void main (String[] args)  {
    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new RMISecurityManager());
    }
    try {
      Context initCtx = new InitialContext(System.getProperties());
      Object obj = initCtx.lookup("hellormiop");
      HelloRemote hello = (HelloRemote)
         javax.rmi.PortableRemoteObject.narrow(obj,
                                               hello.HelloRemote.class);
      System.out.println(hello.sayHello());
      hello.sayThis("Goodbye, Helder!");
      System.out.println(hello.sayHello());

    } catch (Exception e) {
      if (e instanceof RuntimeException)
         throw (RuntimeException)e;
      System.out.println("" + e);
    }
  }
}
```

Obtenção do objeto com associado a"hellormiop" no contexto inicial do serviço de nomes

Não basta fazer cast! O objeto retornado é um objeto CORBA (e não Java) cuja raiz não é java.lang.Object mas org.omg.CORBA.Object

narrow transforma a referência no tipo correto

RMI-IIOP e JNDI



Fonte da ilustração: Mastering EJB2, Ed Roman et al.

Execução

- *Inicie o ORB*

```
> orbd -ORBInitialPort 1900 -ORBInitialHost localhost &
```

- *Rode o servidor*

```
> java hello.rmiop.HelloServer
 -Djava.rmi.server.codebase=file:///aulaj2ee/cap03/build/
 -Djava.security.policy=../lib/rmi.policy
 -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
 -Djava.naming.provider.url=iiop://localhost:1900
Remote object bound to 'hellormiop'.
```

- *Rode o cliente*

```
> java hello.rmiop.HelloClient <mesmas propriedades -D do servidor>
Hello, World!
Goodbye!
```

- *Alternativas para reduzir a quantidade de parâmetros de execução*
  - *Defina as propridades no código ou crie um arquivo rmi.properties (melhor!)*
  - *Use um script shell, .bat ou o Ant*
- *Ant: no subdiretório cap03/, monte a aplicação (semelhante a esta) com*

```
> ant buildrmiop
```

- *Depois, em janelas diferentes, inicie os servidores e cliente digitando* "`ant orbd`", "`ant runrmiopserver`" e "`ant runcorbaclient`"

Objetos serializáveis

- Objetos (parâmetros, tipos de retorno) enviados pela rede via RMI ou RMI-IIOP precisam ser serializáveis
  - **Objeto serializado**: objeto convertido em uma representação binária (tipo BLOB) reversível que preserva informações da classe e estado dos seus dados
  - Serialização representa em único BLOB todo o estado do objeto e de todas as suas dependências, recursivamente
  - Usado como formato "instantâneo" de dados
  - Usado por RMI para passar parâmetros pela rede
- Como criar um objeto serializável (da forma default*)
  - Acrescentar "implements java.io.Serializable" na declaração da classe e marcar os campos não serializáveis com o modificador transient
  - Garantir que todos os campos da classe sejam (1) valores primitivos, (2) objetos serializáveis ou (3) campos declarados com transient

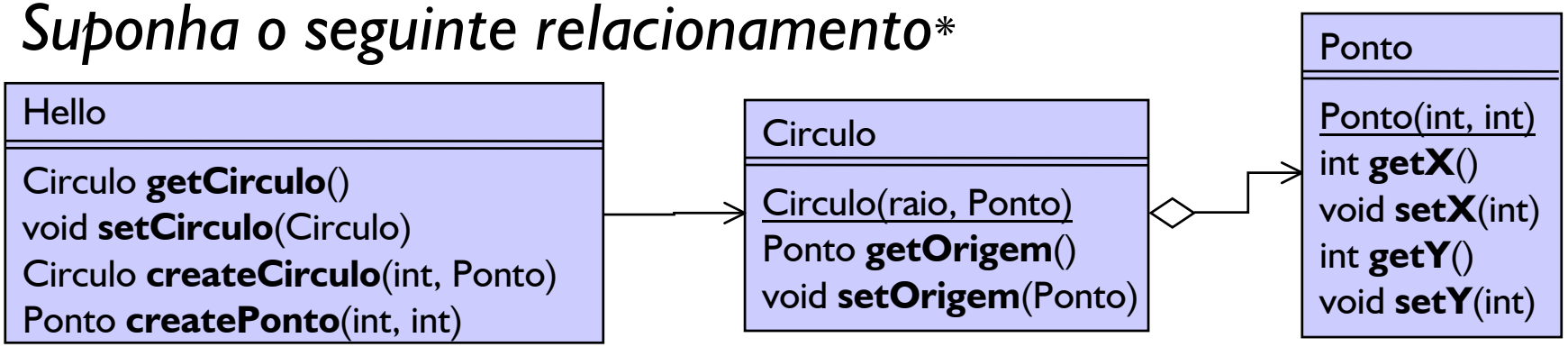
* É possível implementar a serialização de forma personalizada implementando métodos writeObject(), readObject(), writeReplace() e readResolve(). Veja especificação.

Passagem de parâmetros em objetos remotos

- Na chamada de um método remoto, todos os parâmetros são copiados de uma máquina para outra
  - Métodos recebem cópias serializadas dos objetos em passados argumentos
  - Métodos que retornam devolvem uma cópia do objeto
- Diferente da programação local em Java!
  - Quando se passa um objeto como parâmetro de um método a referência é copiada mas o objeto não
  - Quando um método retorna um objeto ele retorna uma referência àquele objeto
- Questões
  - O que acontece quando você chama um método de um objeto retornado por um método remoto?

Passagem de parâmetros local

- *Suponha o seguinte relacionamento**



- *Localmente, com uma referência do tipo Hello, pode-se*
  - *(a) Obter um Circulo*

```
Circulo c1 = hello.getCirculo();
```

Este método chama new Circulo(raio, Ponto)

  - *(b) Trocar o Circulo por outro*

```
hello.setCirculo(hello.createCirculo(50, new Ponto(30, 40));
```

  - *(c) Mudar o objeto Ponto que pertence ao Circulo criado em (b)*

```
Circulo c2 = hello.getCirculo();
c2.setOrigem(hello.createPonto(300, 400));
```

  - *(d) Mudar o valor da coordenada x do ponto criado em (c)*

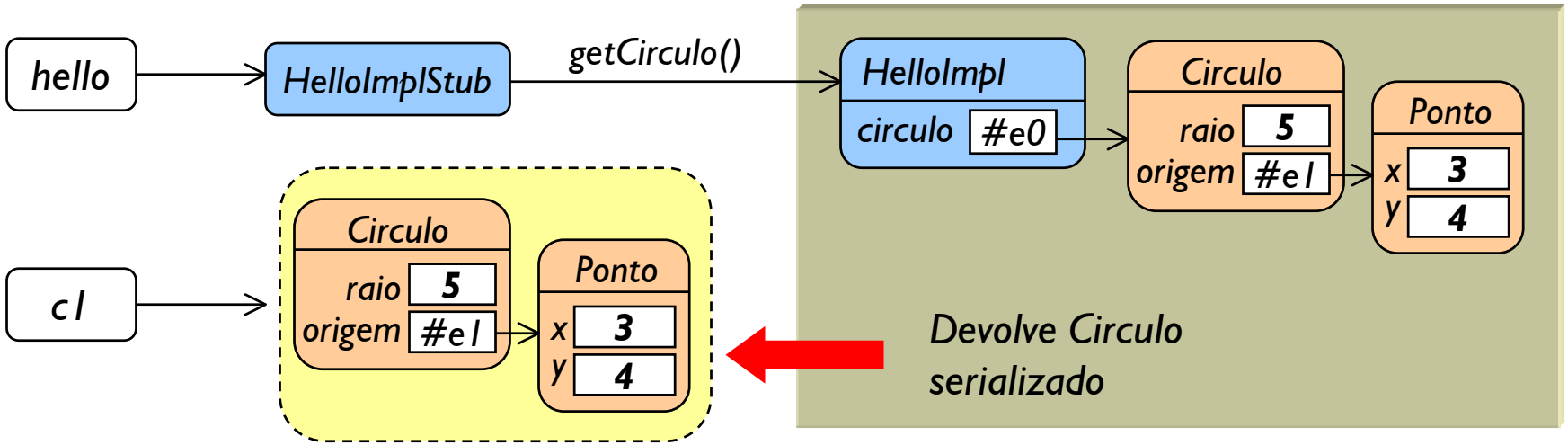
```
c2.getOrigem().setX(3000);
```

... e remotamente?

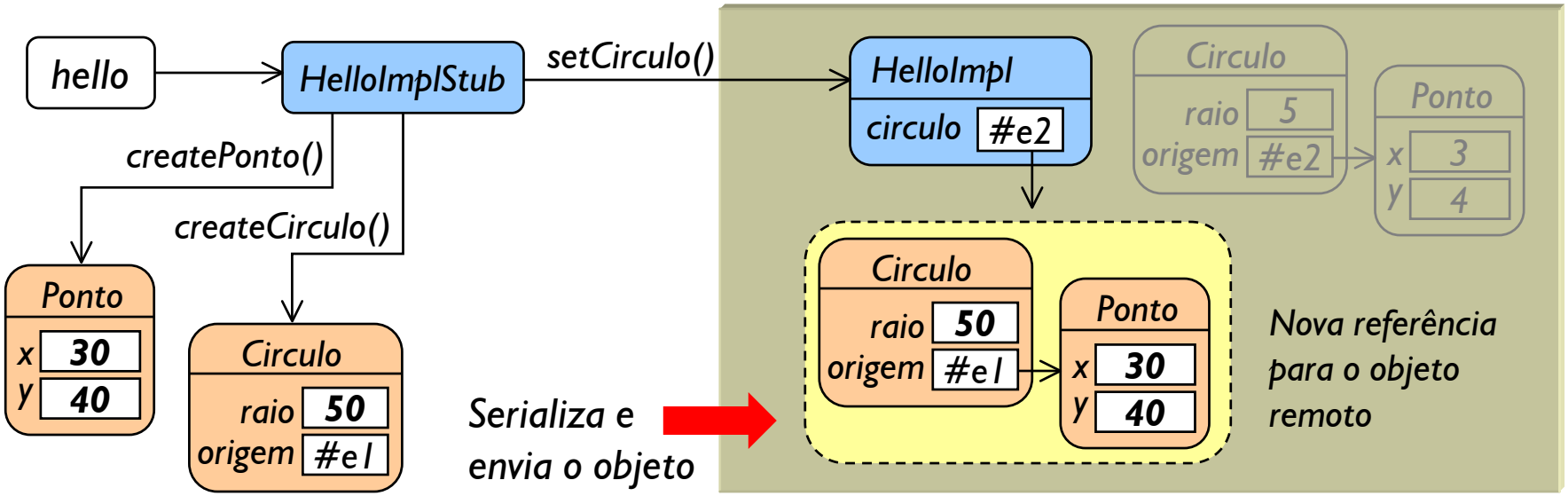
* *veja cap03/src*

Passagem de parâmetros em RMI

a `Circulo c1 = hello.getCirculo();` *// obtém cópia serializada do círculo remoto*

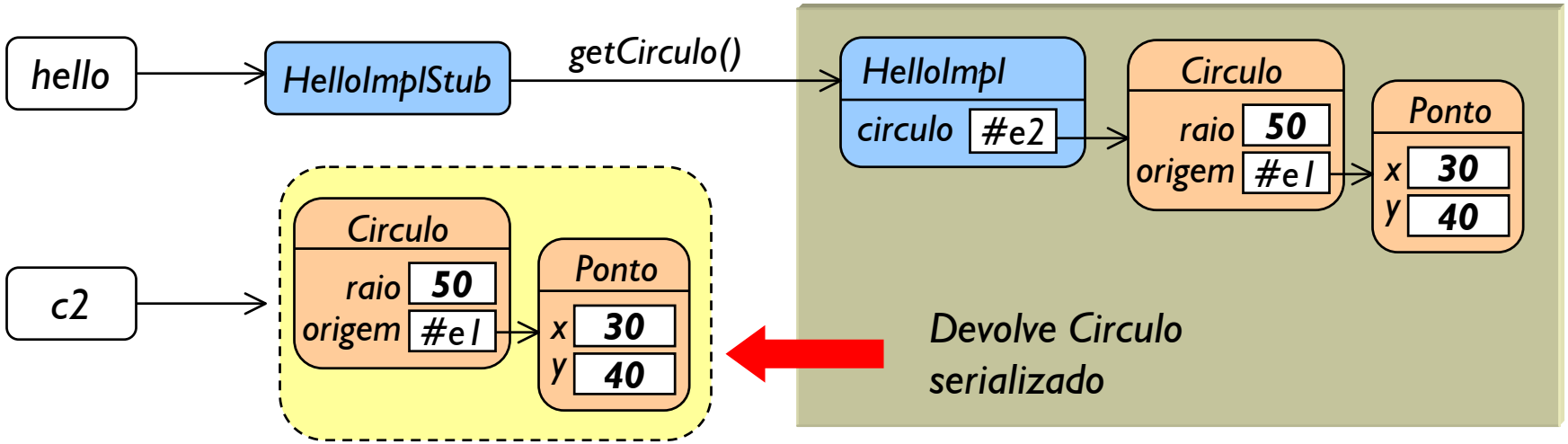


b `hello.setCirculo(hello.createCirculo(50, hello.createPonto(30, 40)));`



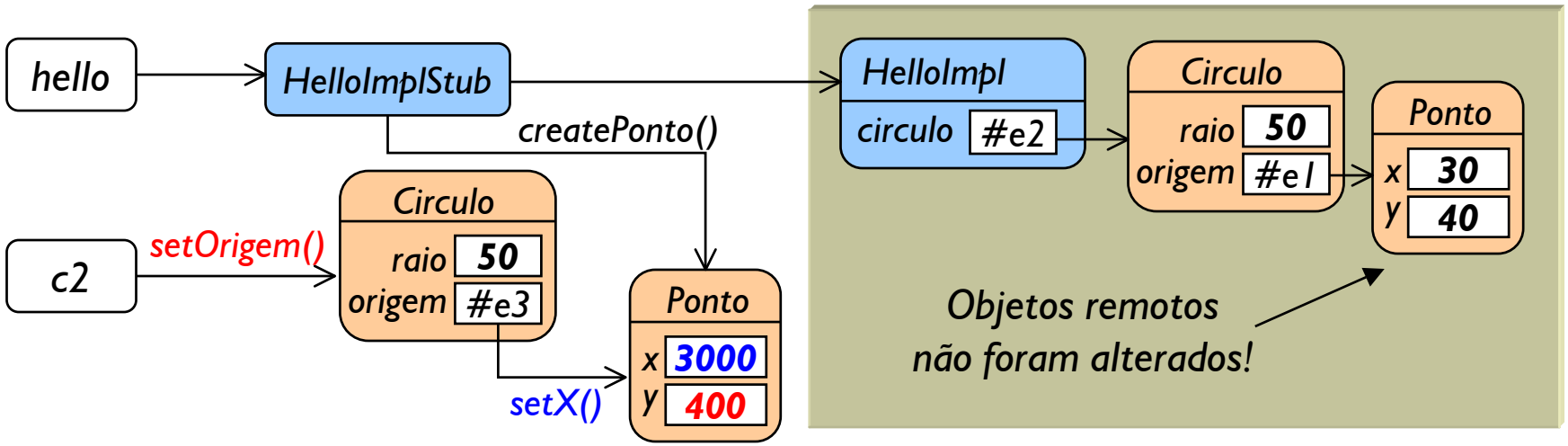
Conseqüências da passagem por valor

a `Circulo c2 = hello.getCirculo();` *// obtém cópia serializada do círculo remoto*



c `c2.setOrigem(hello.createPonto(300, 400));` *// Circulo **local**; Ponto **local**!*

d `c2.getOrigem().setX(3000);` *// altera objeto Ponto **local!***

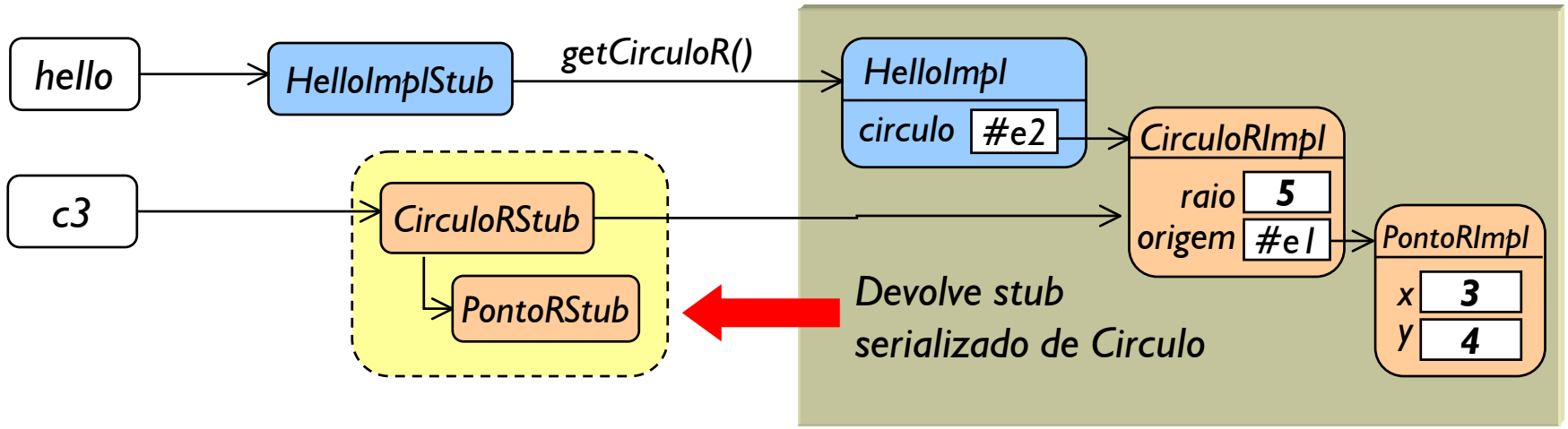


Passagem de parâmetros por referência

- Aplicações RMI sempre passam valores através da rede
  - A referência local de um objeto só tem utilidade local
- Desvantagens
  - Objetos cuja interface é formada pelos seus componentes (alteração nos componentes não é refletida no objeto remoto)
  - Objetos grandes (demora para transferir)
- Solução: passagem por referência
- RMI simula passagem por referência através de stubs
  - Em vez de devolver o objeto serializado, é devolvido um stub que aponta para objeto remoto (objeto devolvido, portanto, precisa ser "objeto remoto", ou seja, implementar java.rmi.Remote)
  - Se cliente altera objeto recebido, stub altera objeto remoto
- Como implementar?
  - Basta que parâmetro ou tipo de retorno seja objeto remoto
  - Objetos remotos são sempre retornados como stubs (referências)

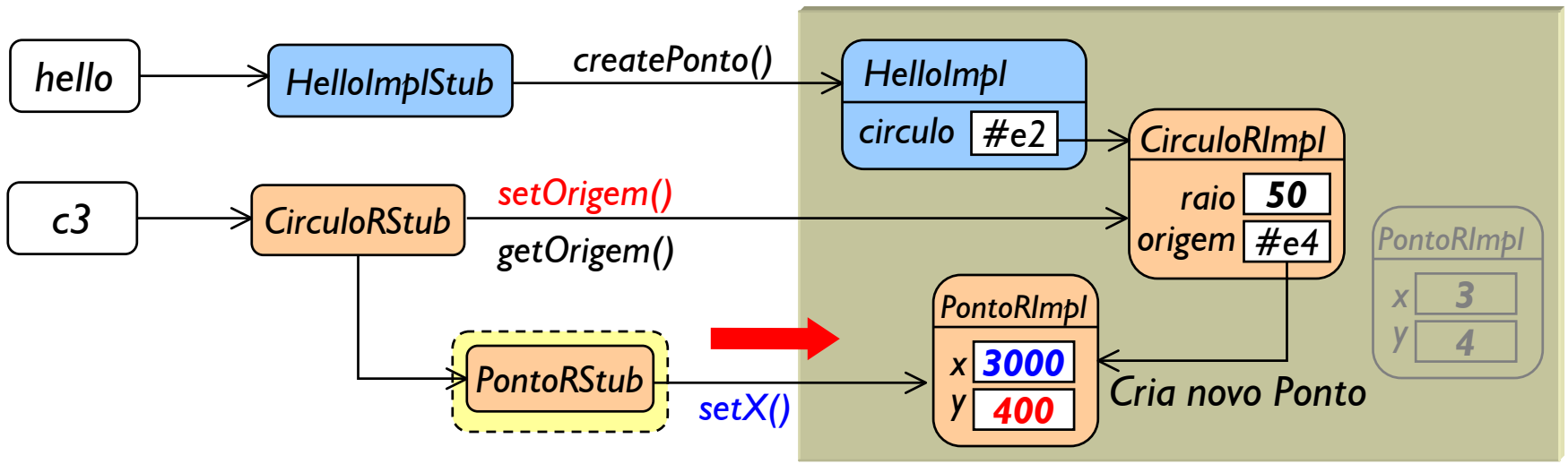
Passagem por referência

a `CirculoR c3 = hello.getCirculoR();` *// obtém referência (stub) para círculo remoto*



c `c2.setOrigem(hello.createPonto(300, 400));` *// CirculoR remoto; PontoR remoto*

d `c2.getOrigem().setX(3000);` *// altera PontoR remoto (que acaba de ser criado)*



Passagem de parâmetros: resumo

- *Quando você passa um objeto como parâmetro de um método, ou quando um método retorna um objeto...*
  - *... em programação Java local*
    - *Referência local (**número** que contém endereço de memória) do objeto é passada*
  - *... em Java RMI (RMI-JRMP ou RMI-IIOP)*
    - *Se tipo de retorno ou parâmetro for objeto remoto (implementa java.rmi.Remote), stub (**objeto** que contém referência remota) é serializado e passado;*
    - *Se parâmetro ou tipo de retorno for objeto serializável mas não remoto (não implementa java.rmi.Remote), uma copia serializada do objeto é passada*

Veja demonstração: Rode, a partir do diretório cap03, `ant build`*, depois, em janelas separadas,* `ant orbd`, `ant runrmiopserver` e `ant runrmiopclient`

Conclusão

- *Java oferece três opções nativas para implementar sistemas de objetos distribuídos*
- *Use RMI sobre JRMP se*
  - *Sua aplicação é 100% Java e sempre vai ser 100% Java*
  - *Você não precisa de um sistema de nomes distribuído e hierárquico*
  - *Os seus objetos nunca mudam de servidor*
  - *Quiser usar recursos exclusivos do RMI (activation, DGC, download)*
- *Use RMI sobre IIOP se você*
  - *Prefere o modelo de desenvolvimento Java RMI (deseja especificar suas interfaces em Java e não em IDL)*
  - *Quer usufruir da melhor escalabilidade do modelo CORBA*
  - *Precisa garantir interoperabilidade com sistemas CORBA*
- *Use Java IDL se você*
  - *Já tem um sistema especificado através de IDLs e precisa criar clientes ou servidores que se comuniquem via IIOP*

Exercício

- *1. Implemente o seguinte objeto como um objeto remoto de uma aplicação RMI-IIOP*

```
public class MensagemImpl {
    private String mensagem = "Inicial";
    public MensagemImpl() {}
    public String getMensagem() {
        return mensagem;
    }
    public void setMensagem(String msg) {
        mensagem = msg;
    }
}
```

  - *a) Escreva a interface Remote*
  - *b) Transforme a classe acima em um objeto remoto*
  - *c) Gere os stubs e skeletons*
  - *d) Escreva um servidor que faça o bind do objeto remoto e um cliente que faça o lookup*
  - *e) Rode o ORBD, o servidor e o cliente*

Fontes

[1] Sun Microsystems. *Java IDL Documentation*. http://java.sun.com/j2se/1.4/docs/guide/idl/
Ponto de partida para tutoriais e especificações sobre Java IDL (parte da documentação do J2SDK 1.4)

[2] Sun Microsystems. *RMI-IIOP Tutorial*. http://java.sun.com/j2se/1.4/docs/guide/rmi-iiop/
Ponto de partida para tutoriais e especificações sobre Java RMI-IIOP (parte da documentação do J2SDK 1.4)

[3] Ed Roman et al. *Mastering EJB 2, Appendixes A and B: RMI-IIOP, JNDI and Java IDL*
http://www.theserverside.com/books/masteringEJB/index.jsp
Contém um breve e objetivo tutorial sobre os três assuntos

[4] Jim Farley. *Java Distributed Computing*. O'Reilly and Associates, 1998. Esta é a primeira edição (procure a segunda). Compara várias estratégias de computação distribuída em Java.

[5] Helder da Rocha, *Análise Comparativa de Dempenho entre Tecnologias Distribuídas Java*. UFPB, 1999, Tese de Mestrado.

[6] Qusay H. Mahmoud Distributed Programming with CORBA (Manning)
http://developer.java.sun.com/developer/Books/corba/ch11.pdf
Breve e objetivo tutorial CORBA

[7] Qusay H. Mahmoud Distributed Java Programming with RMI and CORBA
http://developer.java.sun.com/developer/technicalArticles/RMI/rmi_corba/ Sun, Maio 2002.
Artigo que compara RMI com CORBA e desenvolve uma aplicação que transfere arquivos remotos em RMI e CORBA

helder@argonavis.com.br

argonavis.com.br

Java em Ambientes Distribuídos, 1998
Java RMI x CORBA x RMI-IIOP, 1999
jav500 - Introdução a Java 2 Enterprise Edition, Junho 2002

Fundamentos de JMS

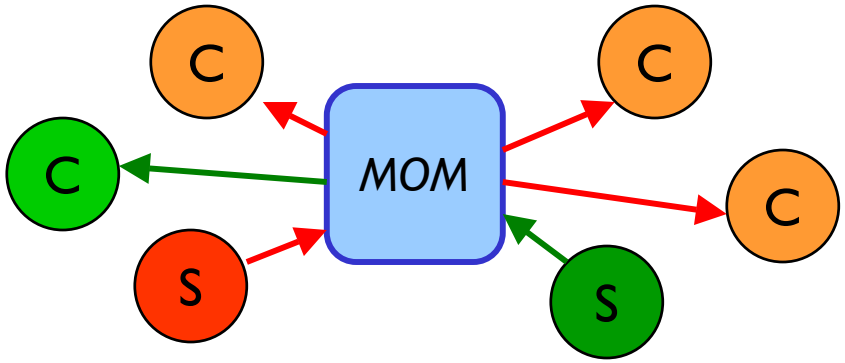
Helder da Rocha
www.argonavis.com.br

Introdução

- *O objetivo deste capítulo é apresentar uma introdução ao modelo de comunicações baseado em mensagens (messaging) e como implementá-lo em aplicações Java usando o Java Message Service (JMS)*
  - *Conceitos fundamentais de messaging e MOMs*
  - *Exemplos de aplicações (estilo "Hello World") para os dois paradigmas de messaging (PTP e pub/sub)*
  - *Como usar o J2EE RI e JBossMQ como provedores JMS*
- *Este capítulo é (em grande parte) baseado no **JMS Tutorial** da Sun, escrito por Kim Haase*
  - *Tutorial é o "livro-texto" para este capítulo. Veja URL no final.*
  - *Os exemplos e exercícios (que foram levemente alterados) serão demonstrados através do serviço JMS da Sun (J2EE Reference Implementation) e JBossMQ*

O que é Messaging

- *Método de comunicação entre componentes ou aplicações*
  - *Arquitetura peer-to-peer com serviço centralizado para repasse de mensagens recebidas e enviadas*
  - *Clientes e servidores enviam e recebem mensagens para canais administrados por serviço central de mensagens (MOM)*



- *Viabiliza comunicação distribuída com acoplamento fraco*
  - *Interface genérica: MOM ignora conteúdo e repassa qualquer coisa. Formato do conteúdo deve ser conhecido pelas partes*
  - *Assíncrona: Comunicação pode ocorrer mesmo que o cliente e servidor não estejam disponíveis ao mesmo tempo*

Message-oriented Middleware (MOM)

- Sistemas de messaging são freqüentemente chamados de **Message-Oriented Middleware (MOM)**
  - Conceito não é novo: já existiam há algum tempo em implementações proprietárias (e incompatíveis)
  - JMS API: solução independente de fabricante para acessar serviços MOM a partir de clientes Java
- Alguns produtos MOM compatíveis com JMS:
  - Open source: JBossMQ, OpenJMS, JORAM
  - IBM MQSeries, IPlanet MQ, Bea WebLogic, HP-MS, Progress SoniqMQ
  - Mais em: java.sun.com/products/jms/

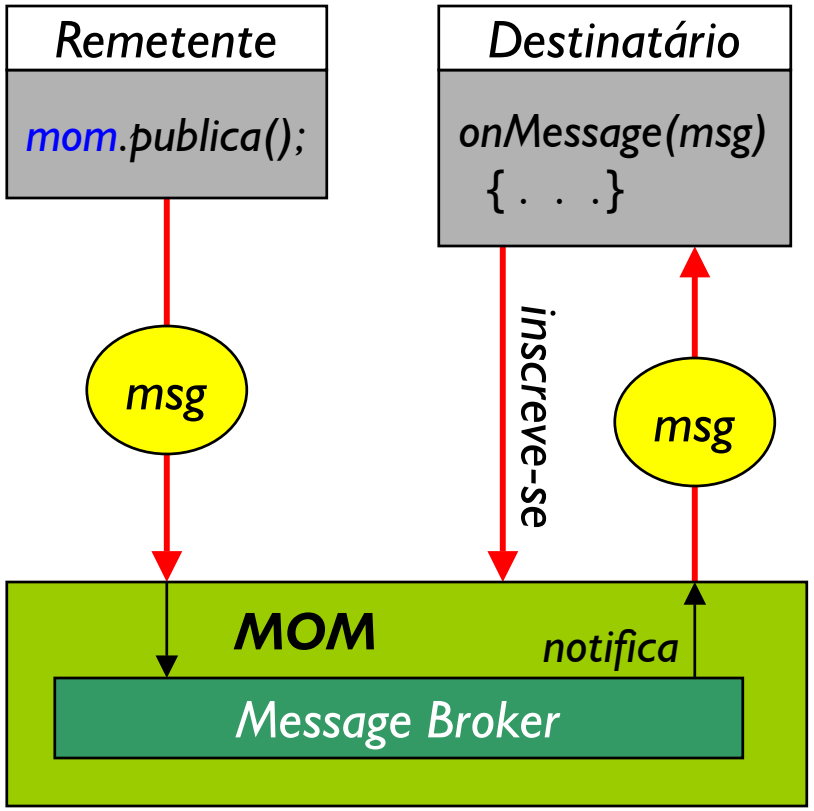
Messaging vs. RMI/RPC vs. E-mail

- Messaging
  - Mensagens são representadas como eventos (que causam número limitado de ações por parte do MOM)
  - Interface genérica (pode ser reutilizada para aplicações diferentes)
  - Arquitetura centralizada (tudo passa pelo MOM)
  - Serviços de diretórios localizam canais de comunicação (destinos)
- RMI/RPC (Corba, Java RMI, etc.)
  - Mensagens são representadas como chamadas para métodos remotos (número ilimitado de ações)
  - Cada aplicação se comunica através de uma interface definida
  - Pode ser descentralizado (rede de ORBs ligados por IIOP)
  - Serviços de diretórios localizam objetos
- E-mail
  - Uma ou ambas as partes podem ser usuários humanos

  ⟹ Messaging é sempre comunicação 100% B2B

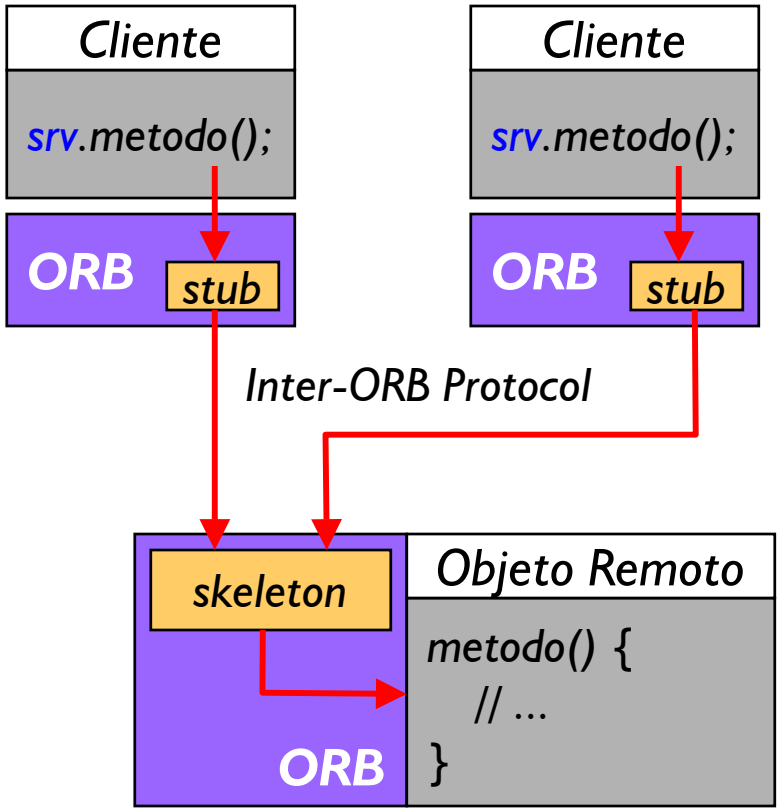
Messaging vs. RMI/RPC

- Sistema de Messaging com paradigma pub/sub



Protocolo comum:
mensagem

- Sistema RMI/RPC (Java RMI ou CORBA)



Protocolo comum:
interface dos objetos

Quando usar um MOM em vez de RMI/RPC?

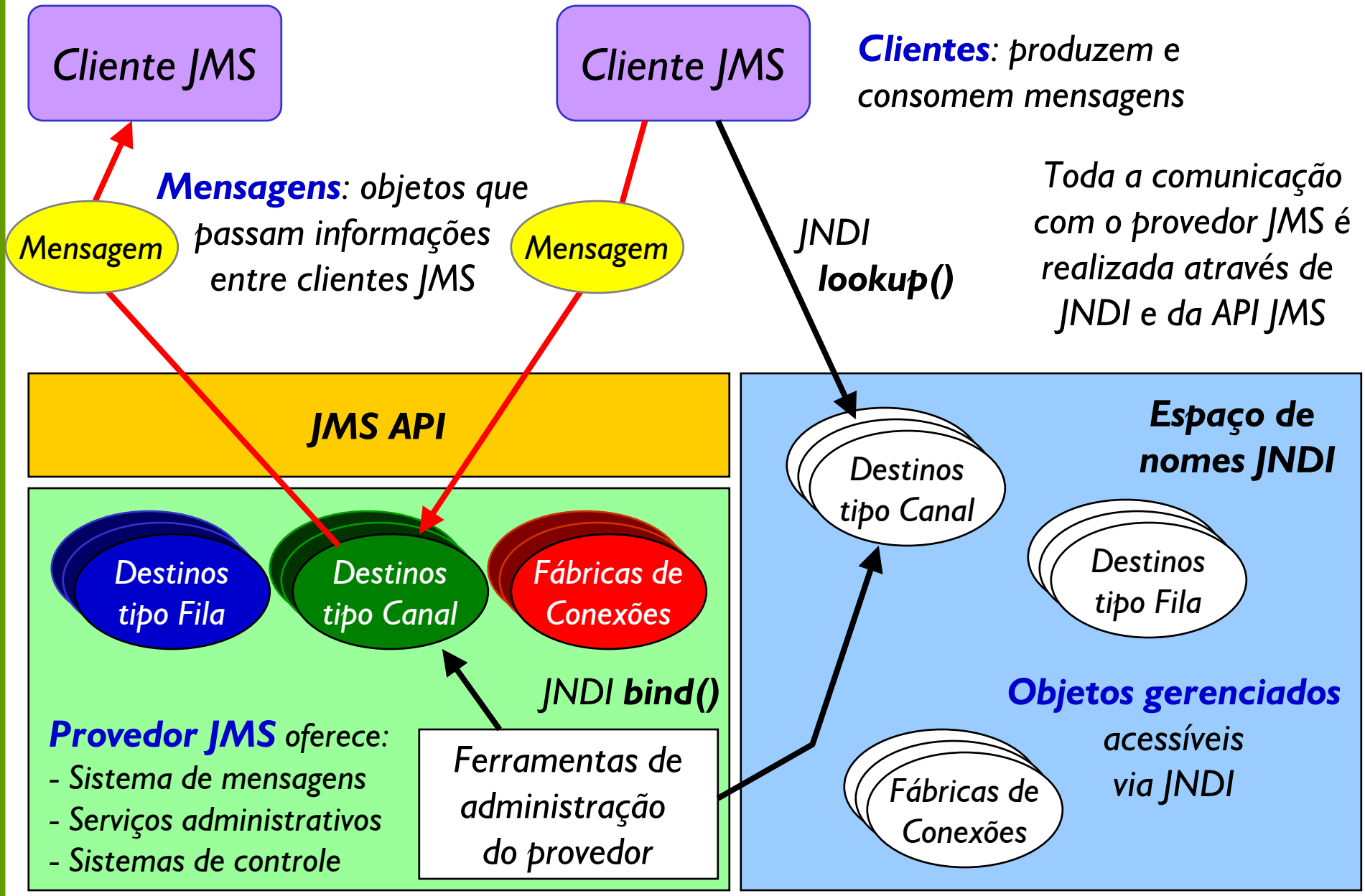
- ... ou, quando decidir por acoplamento mais fraco?
  - Quando a comunicação se baseia mais no formato de mensagens que em interfaces rígidas (componentes não dependem da interface de outros componentes)
  - Quando a disponibilidade dos componentes é imprevisível, mas sua aplicação precisa rodar mesmo que componentes não estejam todos acessíveis
  - Quando for preciso suportar comunicação assíncrona: componente pode enviar informações para outro e continuar a operar mesmo sem receber resposta imediata

➡ **Cenário comum em muitas aplicações B2B!**

Java Message Service

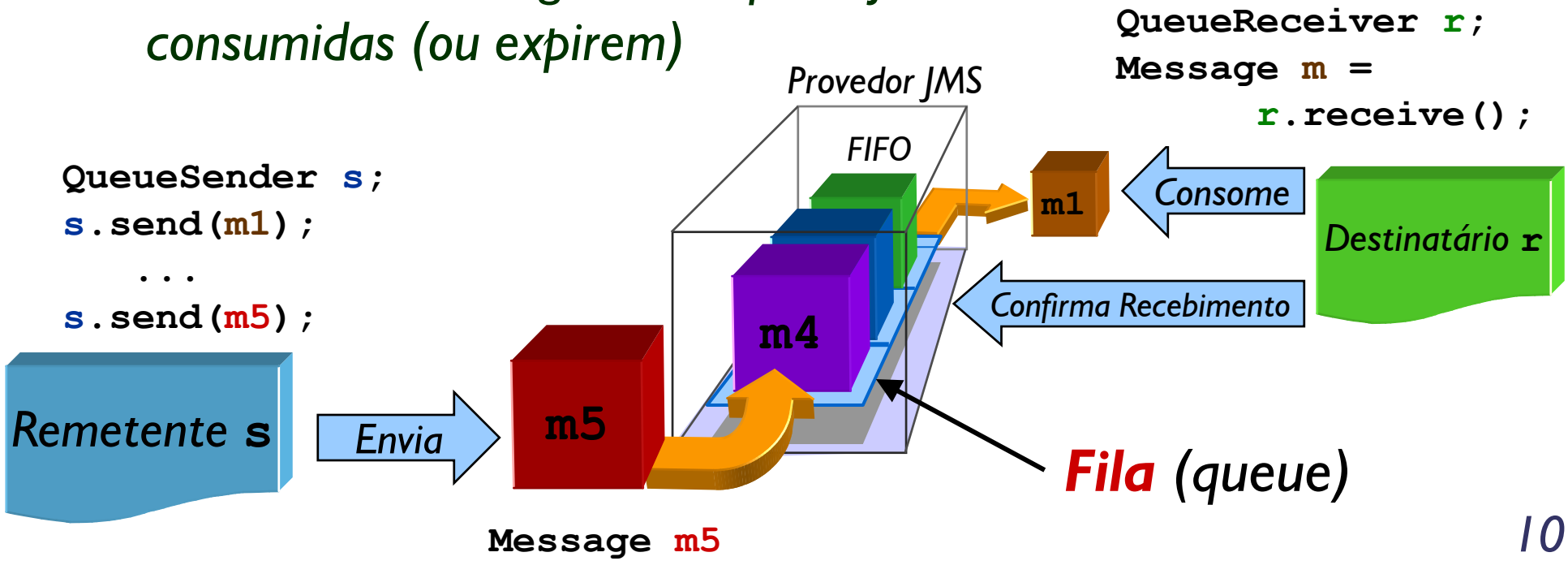
- Interface Java única para unir as MOMs incompatíveis
- **API** que permite que aplicações criem, enviem, recebam e leiam mensagens através de um MOM
  - API consiste principalmente de interfaces (implementadas pelo fabricante do MOM)
  - Parte integral da plataforma J2EE (acrescenta possibilidade de comunicação assíncrona a EJBs)

Arquitetura JMS



Domínio ptp (ponto-a-ponto)

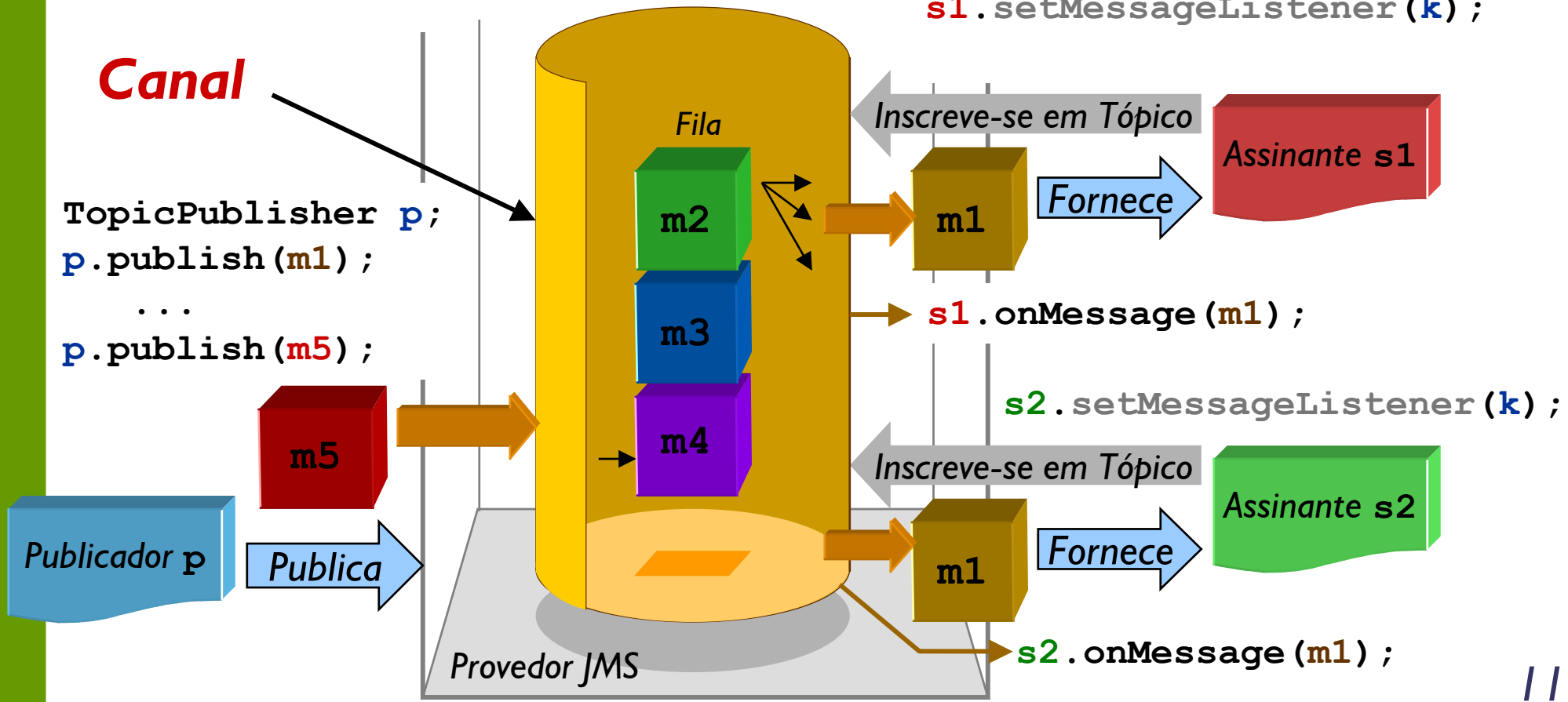
- Baseado no conceito de filas, remetentes e destinatários
  - Um para um: cada mensagem é enviada para uma fila específica e é consumida por um destinatário (que pode ou não estar disponível no momento)
  - Destinatário confirma que a mensagem foi recebida e processada corretamente (acknowledgement)
  - Filas retém mensagens até que sejam consumidas (ou expirem)



Domínio pub/sub (publica/inscreve)

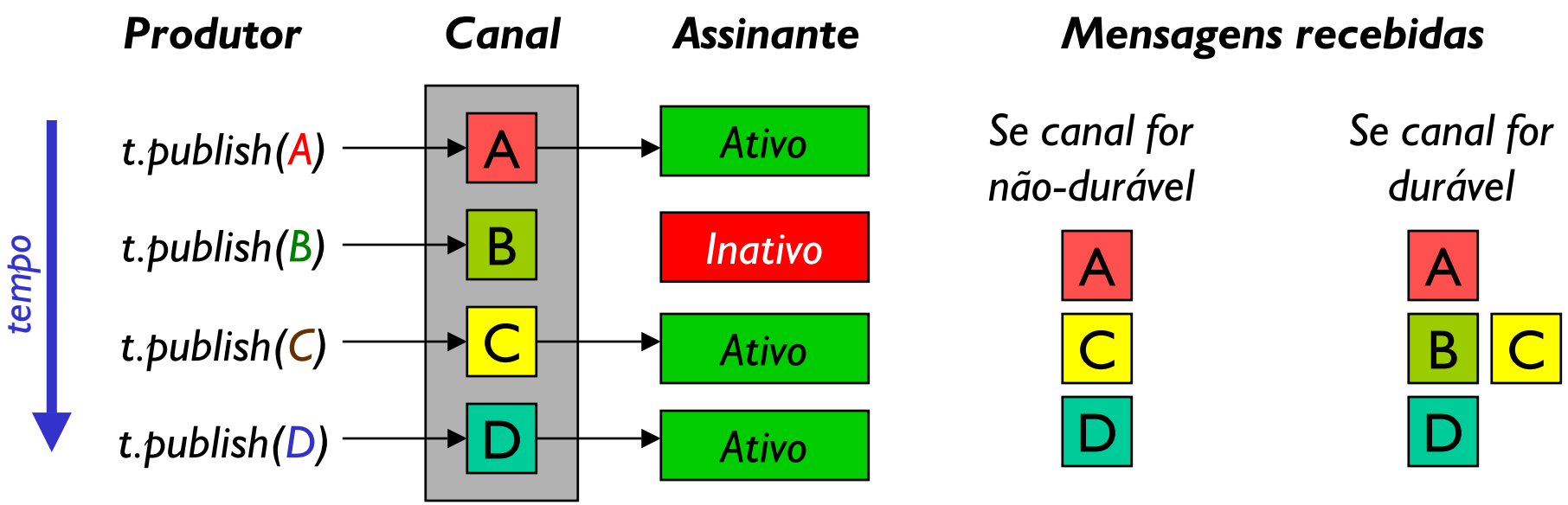
- *Baseado em canais (tópicos)*
  - *Muitos para muitos: mensagens são enviadas a um canal onde todos os assinantes do canal podem retirá-la*
  - *Assinantes recebem notificação*

```
TopicSubscriber s1;
MessageListener k =
   new MessageListener() {
      public void
      onMessage(Message m){...}
   };
s1.setMessageListener(k);
```



Canais duráveis e não-duráveis

- *Cada mensagem enviada a um canal pode ter múltiplos consumidores*
  - *Mensagem permanece disponível até que todos os assinantes a tenham retirado*
  - *Em canais não-duráveis assinante perde as mensagens enviadas nos seus períodos de inatividade*

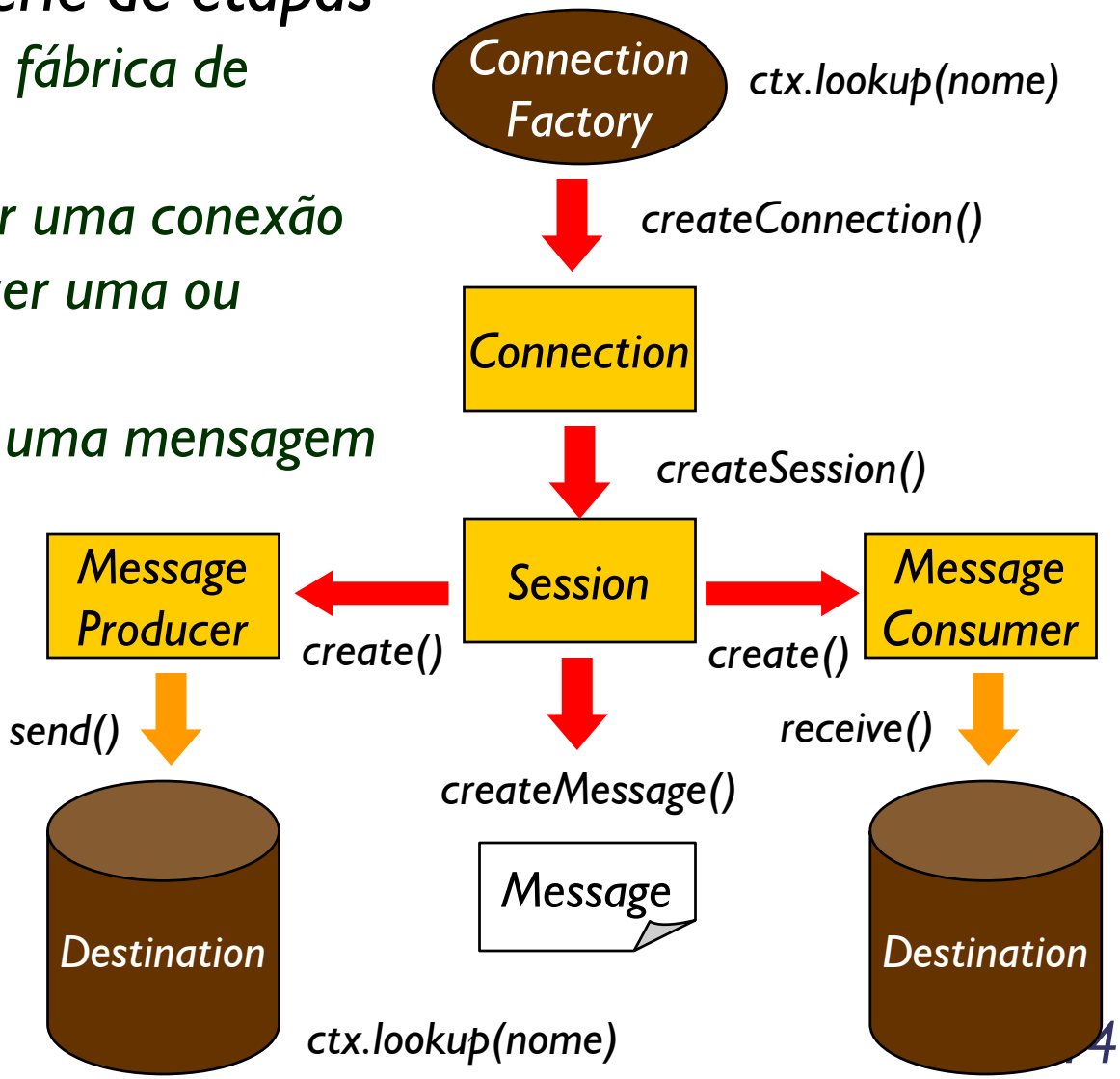


Consumo de mensagens

- *Sistemas de messaging são sempre assíncronos no sentido de que não há dependência quanto ao tempo de envio e recebimento das mensagens*
- *JMS porém permite um tipo de sincronismo*
  - *Pode-se bloquear as operações em um destinatário até que uma determinada mensagem chegue*
- *A especificação JMS, portanto, define que mensagens podem ser consumidas de duas formas*
  - ***Síncrona**: quando o destinatário envia uma chamada receive() e fica a esperar pelo recebimento de mensagens*
  - ***Assíncrona**: o cliente registra-se como ouvinte de mensagens e é notificado quando elas chegam*

Desenvolvimento de aplicações JMS

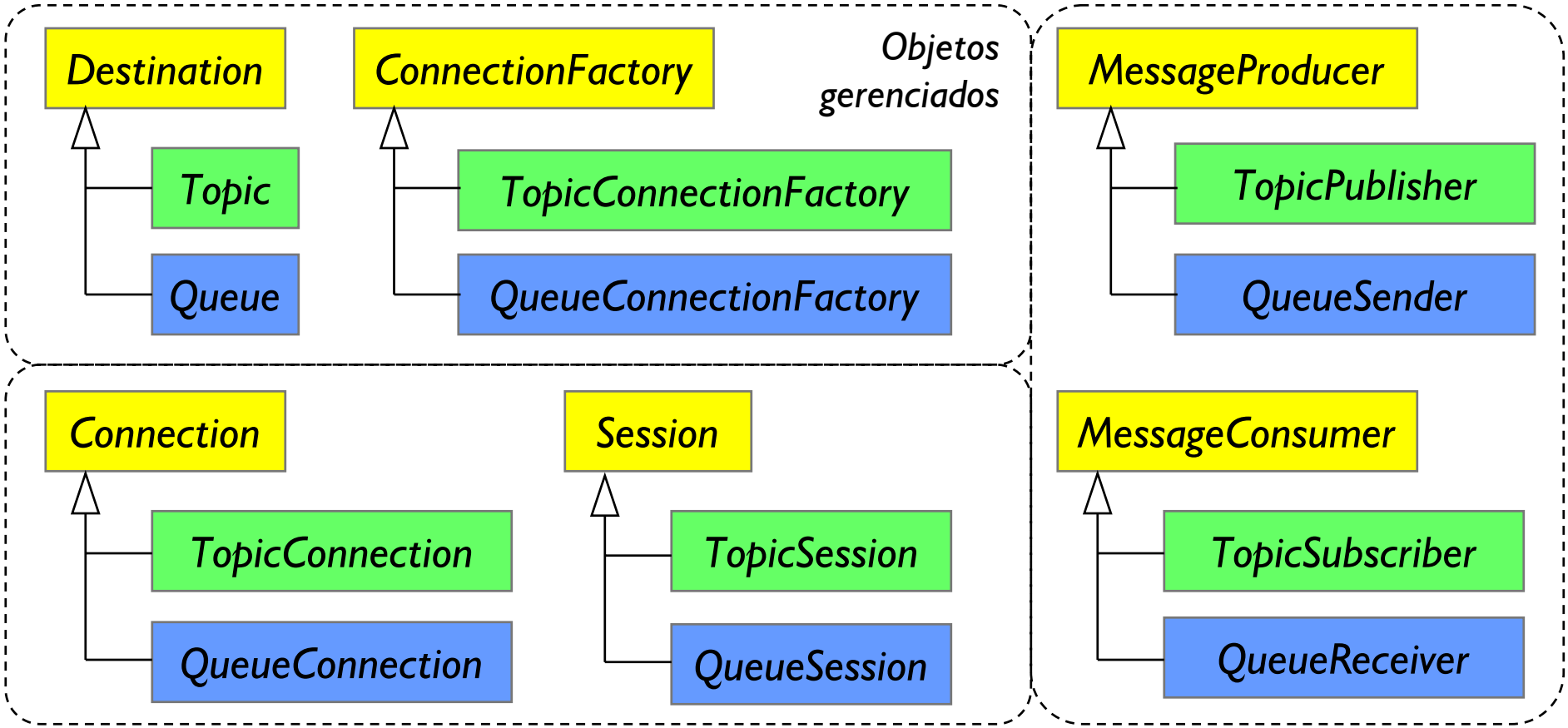
- Para escrever aplicações que enviam ou recebem mensagens é preciso realizar uma série de etapas
  - Obter um destino e uma fábrica de conexões via JNDI
  - Usar a fábrica para obter uma conexão
  - Usar a conexão para obter uma ou mais sessões
  - Usar a sessão para criar uma mensagem
  - Iniciar a sessão
- Depois, pode-se
  - Enviar mensagens
  - Receber mensagens
  - Cadastrar ouvintes para receber mensagens automaticamente



Fonte da ilustração: JMS Tutorial

Interfaces: dois domínios

- *É preciso escolher um domínio*
  - *Para cada domínio há um destino, fábrica de conexões, conexão, sessão, produtor e consumidor*
  - *Interfaces são semelhantes*



- *Domínio pub/sub*
- *Domínio ptp*

Objetos gerenciados

- ***Destinos*** *(filas e tópicos) e* ***fábricas de conexões*** *não podem ser criados através da API*
  - *São criados pelo provedor JMS e ligados a JNDI através de suas ferramentas administrativas. No J2EE-RI:*

```
> j2eeadmin -addJmsDestination queue/MinhaFila queue
```

- *Para usar um objeto gerenciado*
  - *O cliente precisa obtê-lo através de uma consulta (lookup) no serviço de nomes JNDI*
  - *Não há padronização do espaço de nomes (nomes default variam entre fabricantes)*
- *Exemplo (obtenção de um destino do tipo Topic):*

```
String nomeJRI = "jms/Topic"; //Default J2EE RI
String nomeJBoss = "topic/testTopic"; // Default JBossMQ

Context ctx = new InitialContext();
Topic canal = (Topic) ctx.lookup(nomeJBoss);
```

*Há dois tipos de **destinos** JMS*

- ***Filas** (Queue)*
  - *Retêm todas as mensagens que recebem até que sejam retiradas ou expirem*
  - *Para cada mensagem enviada, apenas um cliente pode retirá-la*

```
Queue fila = (Queue) ctx.lookup("jms/Queue");
```

- ***Canais** (Topic)*
  - *Cada canal pode ter vários clientes assinantes*
  - *Cada assinante recebe uma cópia das mensagens enviadas*
  - *Para receber uma mensagem publicada em um canal, clientes precisam já ser assinantes dele antes do envio.*
  - *Em canais "duráveis", assinantes não precisam estar ativos no momento do envio. Retornando, receberão mensagens nao lidas.*

```
Topic canal = (Topic) ctx.lookup("jms/Topic");
```

Fábricas de conexões

- *Antes que se possa*
  - ***enviar** uma mensagem para uma fila,*
  - ***publicar** uma mensagem em um canal,*
  - ***consumir** uma mensagem de uma fila ou*
  - ***fazer uma assinatura** de um canal*

  é preciso obter uma conexão ao provedor JMS
- *Isto é feito através de uma **fábrica de conexões**. Há duas:*
  - *TopicConnectionFactory - para conexões no domínio Topic*
  - *QueueConnectionFactory - para conexões no domínio Queue*
- *É preciso conhecer o nome JNDI*

```
String nomeJRI = "TopicConnectionFactory"; //default J2EE-RI
String nomeJBoss = "ConnectionFactory"; // JBossMQ

Context ctx = new InitialContext();
TopicConnectionFactory factory =
  (TopicConnectionFactory) ctx.lookup(nomeJBoss);
```

Precisa ser definido (geralmente arquivo jndi.properties no CPATH)

Conexões

- *Encapsulam uma conexão virtual com o provedor JMS*
  - *Suportam múltiplas sessões (threads)*
- *Uma vez obtida uma fábrica de conexões, pode-se obter uma conexão*

```
QueueConnection queueCon =
     queueConnectionFactory.createQueueConnection();

TopicConnection topicCon =
     topicConnectionFactory.createTopicConnection();
```

- *Quando a conexão terminar, é preciso fechá-la*
  - *O fechamento fecha todas as seções, produtores e consumidores associados à conexão*

```
queueCon.close();
topicCon.close();
```

Sessões

- *Contexto onde se produz e se consome mensagens*
  - *Criam produtores, consumidores e mensagens*
  - *Processam a execução de ouvintes*
  - *Single-threaded*
- *Podem ser configuradas para definir*
  - *forma de acknowledgement*
  - *uso ou não de transações (tratar uma série de envios/recebimentos como unidade atômica e controlá-la via commit e rollback)*
- *Exemplos*

Sem transações

Confirmação automática após recebimento correto

```
TopicSession topicSession =
    topicCon.createTopicSession(false,
                                Session.AUTO_ACKNOWLEDGE);
```

Sessão controlada por transações

Tratamento de confirmações não especificado

```
QueueSession queueSession =
    queueCon.createQueueSession(true, 0);
```

Produtores de mensagens

- *Objeto criado pela sessão e usado para enviar mensagens para um destino*
  - *QueueSender: domínio ponto-a-ponto*
  - *TopicPublisher: domínio pub/sub*

```
QueueSender sender =
    queueSession.createSender(fila);
TopicPublisher publisher =
    topicSession.createPublisher(canal);
```

- *Uma vez criado o produtor, ele pode ser usado para enviar mensagens*
  - *send() no domínio ponto-a-ponto*

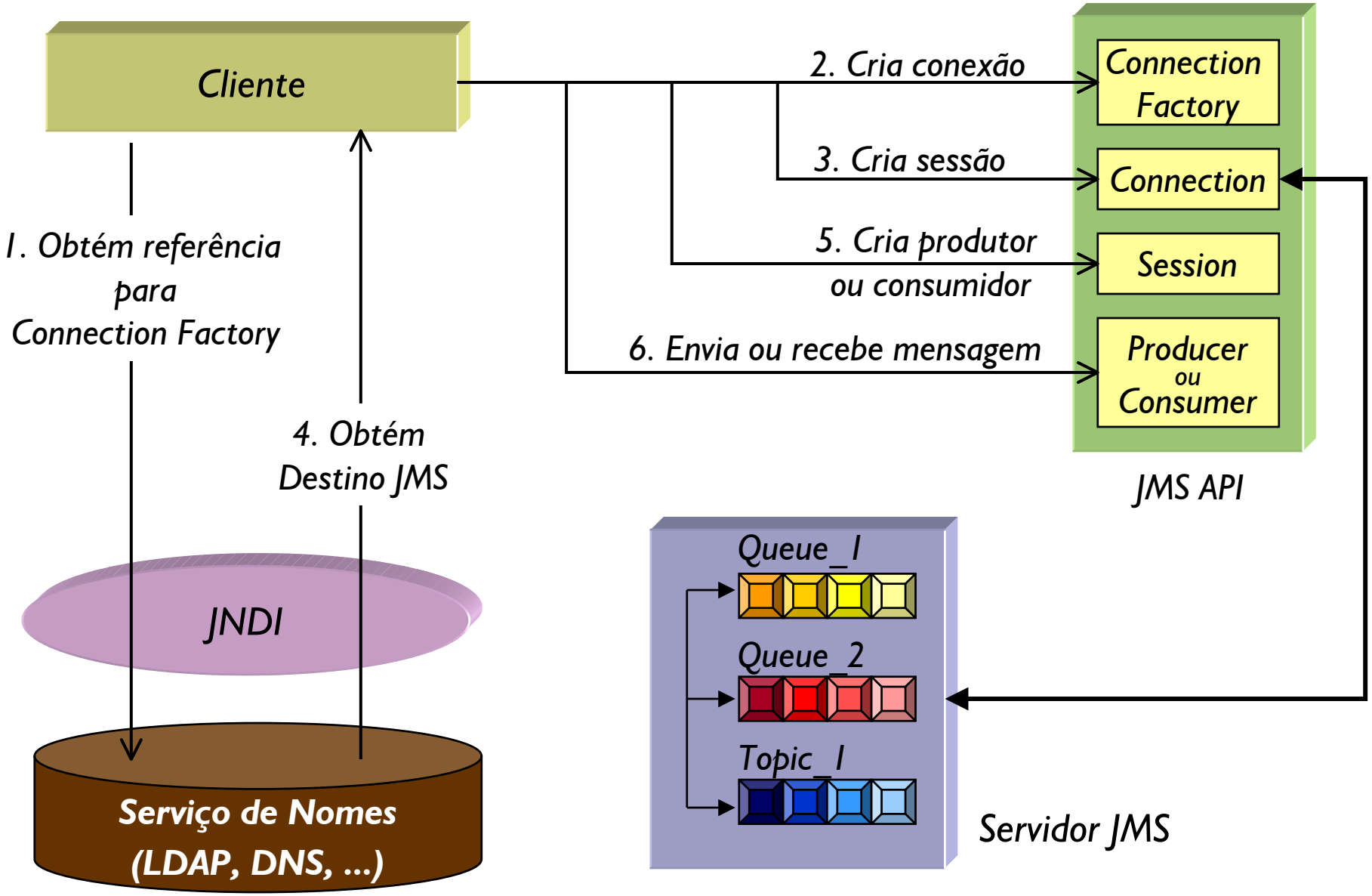
```
sender.send( message );
```

  - *publish() no domínio pub/sub*

```
publisher.publish( message );
```

Durante o envio cliente pode definir ***qualidade do serviço*** *como modo (persistente ou não), prioridade (sobrepõe comportamento FIFO) e tempo de vida (TTL)*

Envio e recebimento de mensagens



Fonte da ilustração: Mastering EJB 2, Cap. 8 (veja referências no final)

Consumidores de mensagens

- *Objeto criado pela sessão e usado para receber mensagens*
  - *QueueReceiver e QueueBrowser: domínio ponto-a-ponto*
  - *TopicSubscriber: domínio pub/sub*

```
QueueReceiver receiver =
     queueSession.createReceiver(fila);
TopicSubscriber subscriber =
     topicSession.createSubscriber(canal);
```

- *É preciso iniciar a conexão antes de começar a consumir:*

```
topicCon.start();
```

- *Depois, pode consumir mensagens de forma síncrona (método é o mesmo para domínios PTP e pub/sub*

```
Message queueMsg = receiver.receive();
Message topicMsg = subscriber.receive(1000);
```

- *Para consumir mensagens de forma assíncrona é preciso criar um MessageListener*

MessageListener

- *Event handler que detecta o recebimento de mensagens*
  - *Para usar, implemente MessageListener e seu método onMessage():*

```
public class MyListener implements MessageListener {
    public void onMessage(Message msg) {
        TextMessage txtMsg = (TextMessage) msg;
        System.out.println( "Mensagem recebida: " +
                            txtMsg.getText() )
    }
}
```

- *Método onMessage() não deve deixar escapar exceções*
  - *Código acima deve estar em um bloco try-catch*
- *Para que objeto seja notificado, é preciso registrá-lo em um QueueReceiver ou TopicSubscriber*

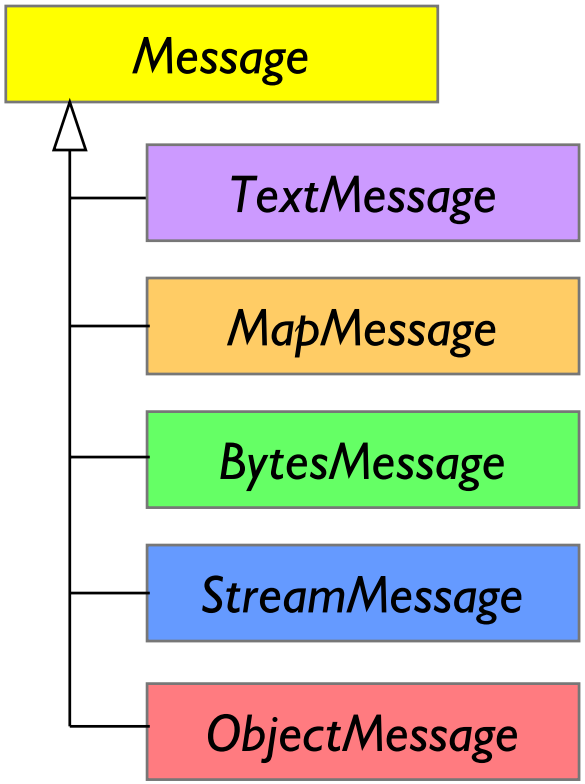
```
subscriber.setMessageListener( new MyListener() );
topicCon.start();  // iniciar a conexão
```

Mensagens

- Mensagens são compostas de três partes
  - **Propriedades** (opcionais): pares nome/valor (nomes e valores arbitrários definidos pela aplicação); contém tipos primitivos Java (int, boolean, etc.) ou String.
  - **Cabeçalhos**: propriedades com nomes e tipos de valores padrão definidos na especificação JMS
  - **Corpo**: conteúdo que não pode ser representado através de propriedades
- Os tipos de mensagem correspondem a formatos de dados armazenados no corpo de mensagem
  - Texto, objetos serializados, bytes, valores primitivos, etc.
- Mensagens são criadas a partir de uma Session

Seis tipos de mensagens

- *Message*
  - *Mensagem genérica sem corpo (contendo apenas cabeçalho e possíveis propriedades)*
- *TextMessage*
  - *Objeto do tipo String (ex: conteúdo XML)*
- *MapMessage*
  - *Conjunto de pares nome/valor onde nomes são Strings e valores são tipos primitivos*
- *BytesMessage*
  - *Stream de bytes não interpretados*
- *StreamMessage*
  - *Seqüência de tipos primitivos Java*
- *ObjectMessage*
  - *Objeto Java serializado*



Criação de mensagens

- *Para cada tipo de mensagem, Session fornece método create()*
  - *createMessage(), createTextMessage(), createBytesMessage(), createObjectMessage(), createMapMessage(), createStreamMessage()*

```
TextMessage message =
      queueSession.createTextMessage();
message.setText(msg_text); // msg_text é String
sender.send(message);
```

- *Após receber uma mensagem, via receive() ou onMessage(), é preciso fazer o cast para ter acesso aos métodos específicos de cada tipo de mensagem*

```
Message m = receiver.receive();
if (m instanceof TextMessage) {
  TextMessage message = (TextMessage) m;
  System.out.println("Recebido: " + message.getText());
}
```

Exceções

- A maior parte das operações JMS potencialmente causam *javax.jms.JMSException*
- Há várias sub-classes de JMSException para tratar situãções específicas
  - Veja documentação da API ou JMS Tutorial (3.7) para mais detalhes

Exemplos de aplicações

- Os exemplos estão no capítulo 4 do JMS Tutorial
  - Use-o como referência para configurar o ambiente e executar as aplicações no J2EE RI
- As versões em cap04/src/ estão levemente alteradas
  - Todas estão nos pacotes jmstut.topic ou jmstut.queue
  - Use 'ant' para compilar
  - Use 'ant run' ou 'ant runjboss' para executar
- Aplicação ponto-a-ponto (no J2EE RI)

```
> ant run -Dclass=jmstut.queue.SimpleQueueSender
          -Ddestino=jms/Queue
          -Dqte=5
> ant run -Dclass=jmstut.queue.SimpleQueueReceiver
          -Ddestino=jms/Queue
```

- Aplicação pub/sub
  - Mesma coisa! Mude apenas a classe e o destino

Configuração do ambiente

- Sun J2EE Reference Implementation
  - Consulte o JMS Tutorial, capítulo 4.
- JBossMQ
  - Inicie o servidor
  - Rode o ant (slide anterior) usando a target 'runjboss'
- Nomes JNDI para destinos default
  - queue/testQueue
  - topic/testTopic
- Nomes JNDI para fábricas default
  - ConnectionFactory (usado para os dois domínios)
- Veja o build.xml e build.properties para detalhes sobre configuração
- Quer criar novos tópicos, fábricas?
  - http://localhost:8082

Conclusão

- Você já conhece os "fundamentos" do JMS
- Veja mais no JMS Tutorial
  - Aplicações robustas, confiáveis, usando transações, tempo de expiração, controle de persistência, níveis de prioridade, seleção usando filtros SQL em propriedades
  - Exemplos de aplicações J2EE usando JMS
  - Exemplos de aplicações usando dois servidores J2EE
- Voltaremos a falar de JMS mais adiante no Capítulo 12: Message-driven beans
  - EJBs assíncronos assumindo o papel de MessageListeners
  - Transações distribuídas
  - Interação com clientes Web

Exercícios

- *1. Instale as aplicações mostradas e execute-as*
  - *No J2EE RI*
  - *No JBossMQ*
  - *No JORAM (veja diretório software/jms)*

Fontes

[1] *Kim Haase. The JMS Tutorial. Sun Microsystems, 2002*
http://java.sun.com/products/jms/tutorial/ Este capítulo é baseado no JMS Tutorial.

[2] *Sun Microsystems. The Java Message Service Specification*
http://www.javasoft.com/products/jms/docs.html

[3] *Peter Antman. JBoss and JMS. JBoss User's Manual, Chapter 8.*
http://www.jboss.org/online-manual/HTML/ch08.html Contém breve tutorial sobre JMS

[4] *Todd SundSted. Messaging helps move Java into the Enterprise. JavaWorld, Jan 1999.*
http://www.javaworld.com/jw-01-1999/jw-01-jms.html
Introdução aos conceitos fundamentais de sistemas de mensagens.

[5] *Todd SundSted. Messaging makes its move, Parts I & II. JavaWorld, Fev e Mar 1999.*
http://www.javaworld.com/javaworld/jw-02-1999/jw-02-howto_p.html
Série de dois artigos mostrando como usar JMS.

[6] *Gordon Van Huizen. JMS: An infrastructure for XML-based B2B communication*
http://www.javaworld.com/javaworld/jw-02-2000/jw-02-jmsxml.html JavaWorld, 2000.
Introdução a JMS motivada pelo seu potencial para soluções B2B com XML

[7] *Michael Schoffner. Write your own MOM. JavaWorld, 1998.*
http://www.javaworld.com/javaworld/jw-05-1998/jw-05-step.html

[8] *Ed Roman et al. Mastering EJB 2, Chapter 8: Message Drive Beans*
http://www.theserverside.com/books/masteringEJB/index.jsp
Contém um breve e objetivo tutorial sobre JMS no início do capítulo

helder@argonavis.com.br

argonavis.com.br

Minicurso de JMS, 2002

Componentes e Clientes Web

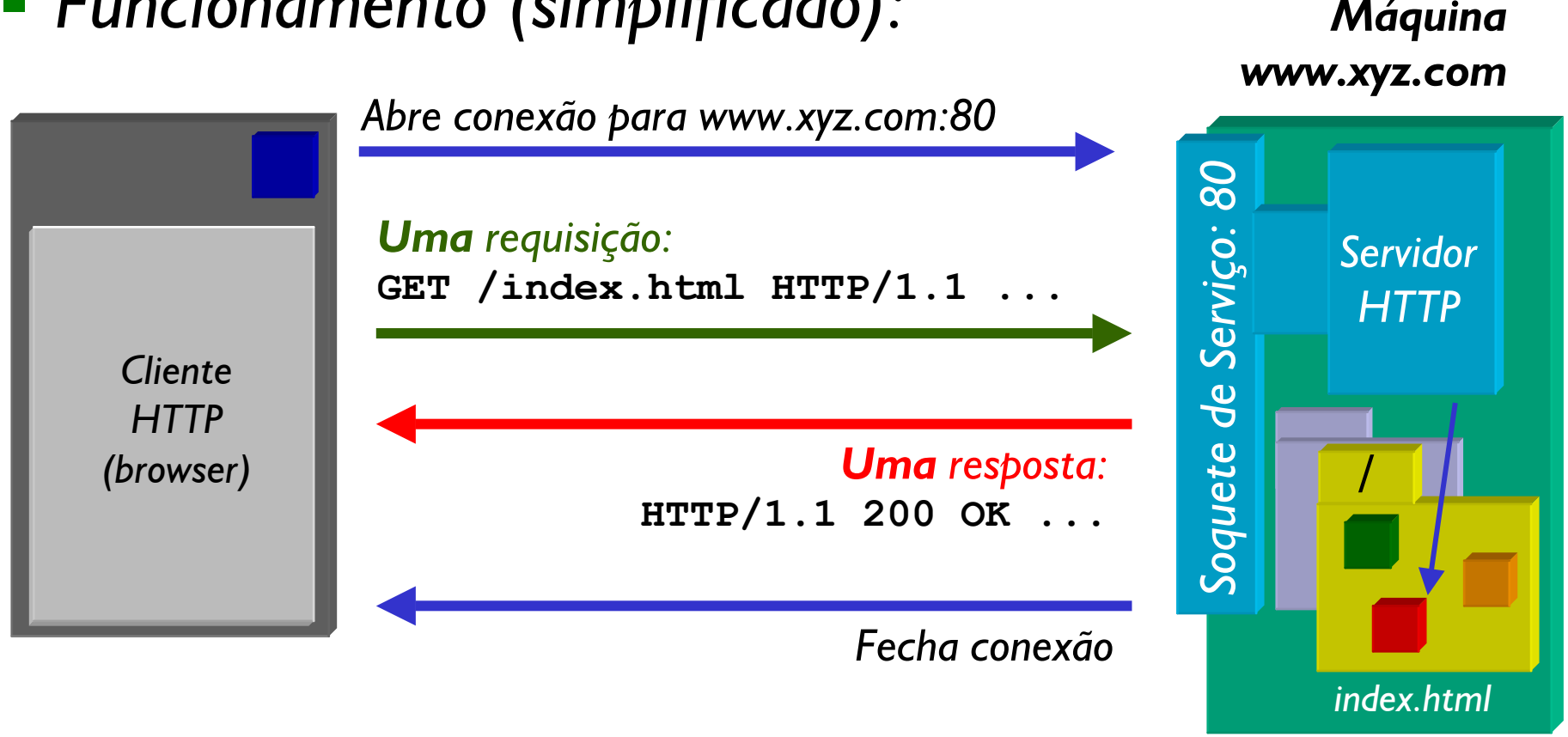
Helder da Rocha
www.argonavis.com.br

Sumário

- *Conceitos essenciais sobre a plataforma Web*
  - *Fundamentos de HTTP: métodos, respostas, requisições*
  - *Tecnologias lado-servidor: CGI, APIs e scripts*
  - *Cookies*
- *Aplicações Web em Java*
  - *Componentes Web*
  - *Servlet containers e conectores*
  - *Jakarta-Tomcat (implementação de referência)*
- *Web application archives (WAR)*
  - *Estrutura de uma aplicação Web*
  - *Arquivos WAR*
  - *Configuração e instalação de aplicações Web (web.xml)*
- *Integração J2EE*
  - *Inclusão de WARs em EARs e implantação no JBoss*

A plataforma Web

- *Baseada em HTTP (RFC 2068)*
  - *Protocolo simples de transferência de arquivos*
  - *Sem estado (não mantém sessão aberta)*
- *Funcionamento (simplificado):*



Cliente e servidor HTTP

- Servidor HTTP
  - Gerencia sistema virtual de arquivos e diretórios
  - Mapeia pastas do sistema de arquivos local (ex: c:\htdocs) a diretórios virtuais (ex: /) acessíveis remotamente (notação de URI)
- Papel do servidor HTTP
  - Interpretar requisições HTTP do cliente (métodos GET, POST, ...)
  - Devolver resposta HTTP à saída padrão (código de resposta 200, 404, etc., cabeçalho RFC 822* e dados
- Papel do cliente HTTP
  - Enviar requisições HTTP (GET, POST, HEAD, ...) a um servidor. Requisições contém URI do recurso remoto, cabeçalhos RFC 822 e opcionalmente, dados (se método HTTP for POST)
  - Processar respostas HTTP recebidas (interpretar cabeçalhos, identificar tipo de dados, interpretar dados ou repassá-los.

* Padrão Internet para construção de cabeçalhos de e-mail

Principais métodos HTTP (requisição)

- ***GET*** *- pede ao servidor um arquivo (informado sua URI) absoluta (relativa à raiz do servidor)*

```
GET <uri> <protocolo>/<versão>
<Cabeçalhos HTTP>: <valores> (RFC 822)
<linha em branco>
```

  - *GET pode enviar dados através da URI (tamanho limitado)*

```
<uri>?dados
```

  - *Método HEAD é idêntico ao GET mas servidor não devolve página (devolve apenas o cabeçalho)*

- ***POST*** *- envia dados ao servidor (como fluxo de bytes)*

```
POST <uri> <protocolo>/<versão>
<Cabeçalhos HTTP>: <valores>
<linha em branco>
<dados>
```

Cabeçalhos HTTP

- *Na **requisição**, passam informações do cliente ao servidor*
  - *Fabricante e nome do browser, data da cópia em cache, cookies válidos para o domínio e caminho da URL da requisição, etc.*
- *Exemplos:*

```
User-Agent: Mozilla 5.5 (Compatible; MSIE 6.0; MacOS X)
If-Modified-Since: Thu, 23-Jun-1999 00:34:25 GMT
Cookies: id=344; user=Jack; flv=yes; mis=no
```

- *Na **resposta**: passam informações do servidor ao cliente*
  - *Tipo de dados do conteúdo (text/xml, image/gif) e tamanho, cookies que devem ser criados. endereço para redirecionamento, etc.*
- *Exemplos:*

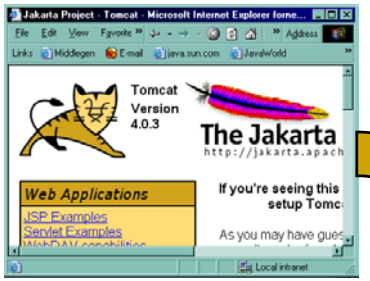
```
Content-type: text/html; charset-iso-8859-1
Refresh: 15; url=/pags/novaPag.html
Content-length: 246
Set-Cookie: nome=valor; expires=Mon, 12-03-2001 13:03:00 GMT
```

Comunicação HTTP: detalhes

- *1. Página HTML*

```
<img src="tomcat.gif" />
```

Interpreta HTML



Gera requisição GET

- *2. Requisição: browser solicita imagem*

```
GET tomcat.gif HTTP/1.1
User-Agent: Mozilla 6.0 [en] (Windows 95; I)
Cookies: querty=uiop; SessionID=D236S11943245
→
```

Linha em branco termina cabeçalhos

- *3. Resposta: servidor devolve cabeçalho + stream*

```
HTTP 1.1 200 OK
Server: Apache 1.32
Date: Friday, August 13, 2003 03:12:56 GMT-03
Content-type: image/gif
Content-length: 23779
→
!#GIF89~¾ 7
.55.a  6¤Ü4  ...
```

tomcat.gif

Tecnologias lado-servidor

- ***Estendem*** *as funções básicas de servidor HTTP:*
  - *CGI - Common Gateway Interface*
  - *APIs: ISAPI, NSAPI, Apache API, Servlet API, ...*
  - *Scripts: ASP, JSP, LiveWire (SSJS), Cold Fusion, PHP, ...*
- *Rodam do lado do servidor, portanto, não dependem de suporte por parte dos browsers*
  - *browsers fornecem apenas a interface do usuário*
- *Interceptam o curso normal da comunicação*
  - *Recebem dados via requisições HTTP (GET e POST)*
  - *Devolvem dados através de respostas HTTP*

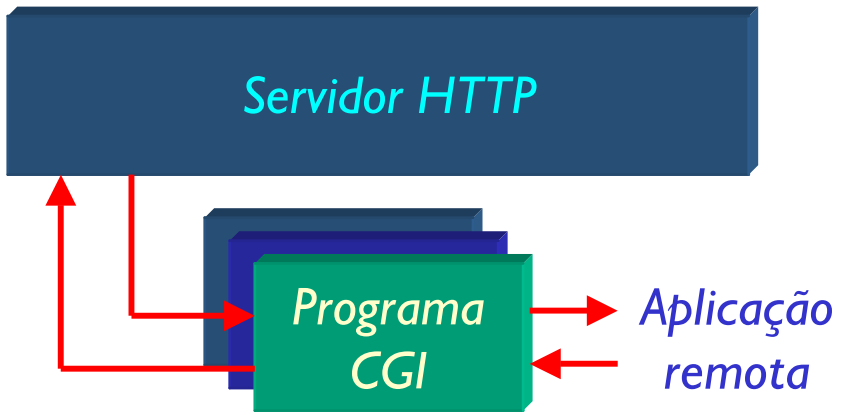
CGI - Common Gateway Interface

- ***Especificação*** *que determina como construir uma aplicação que será executada pelo servidor Web*
- *Programas CGI podem ser escritos em qualquer linguagem de programação. A especificação limita-se a determinar os formatos de entrada e saída dos dados (HTTP).*
- *O que interessa é que o programa seja capaz de*
  - *Obter dados de entrada a partir de uma requisição HTTP*
  - *Gerar uma resposta HTTP incluindo os dados e parte do cabeçalho*

- *Escopo: camada do servidor*
  - *Não requer quaisquer funções adicionais do cliente ou do HTTP*

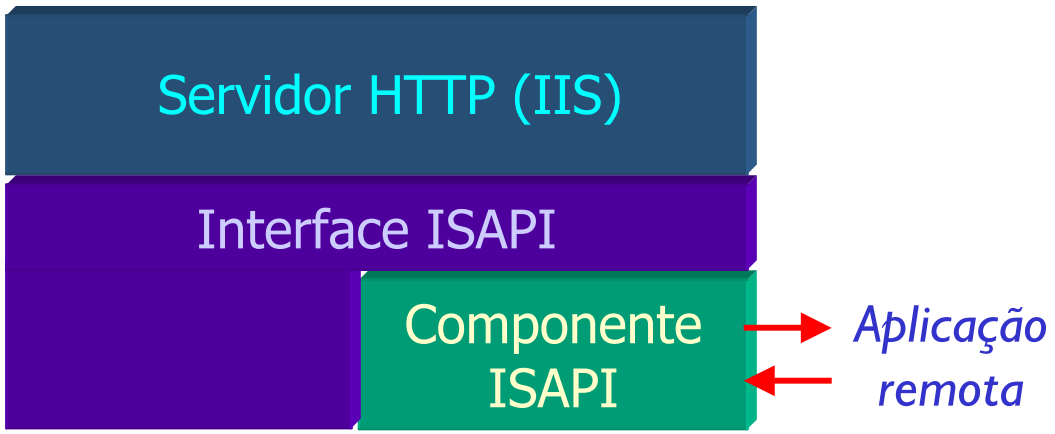
CGI é prático... Mas ineficiente!

- A interface CGI requer que o servidor sempre execute um programa
  - Um novo processo do S.O. rodando o programa CGI é criado para cada cliente remoto que o requisita.
  - Novos processos consomem muitos recursos, portanto, o desempenho do servidor diminui por cliente conectado.
- CGI roda como um processo externo, logo, não tem acesso a recursos do servidor
  - A comunicação com o servidor resume-se à entrada e saída.
  - É difícil o compartilhamento de dados entre processos



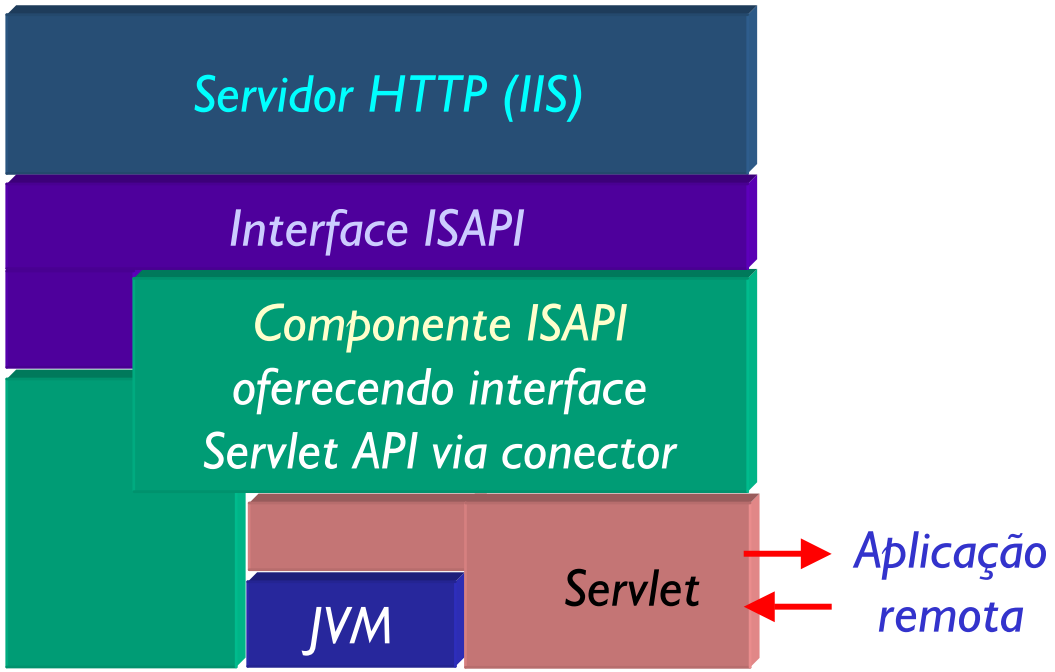
APIs do servidor

- *Podem substituir totalmente o CGI, com vantagens:*
  - *Toda a funcionalidade do servidor pode ser usada*
  - *Múltiplos clientes em processos internos (threads)*
  - *Muito mais rápidas e eficientes (menos overhead)*
- *Desvantagens:*
  - *Em geral dependem de plataforma, fabricante e linguagem*
  - *Soluções proprietárias*
- *Exemplos*
  - *ISAPI (Microsoft)*
  - *NSAPI (Netscape)*
  - *Apache Server API*
  - *??SAPI*



Servlet API

- *API independente de plataforma e praticamente independente de fabricante*
- *Componentes são escritos em Java e se chamam servlets*
- *Como os componentes SAPI proprietários, rodam dentro do servidor, mas através de uma Máquina Virtual Java*
- *Disponível como 'plug-in' ou conector para servidores que não o suportam diretamente*
  - *Desenho ao lado mostra solução antiga de conexão com IIS*
- *Nativo em servidores Sun, IBM, ...*



Vantagens dos servlets...

- ... sobre CGI
  - Rodam como parte do servidor (cada nova requisição inicia um novo thread mas não um novo processo)
  - Mais integrados ao servidor: mais facilidade para compartilhar informações, recuperar e decodificar dados enviados pelo cliente, etc.
- ... sobre APIs proprietárias
  - Não dependem de único servidor ou sistema operacional
  - Têm toda a API Java à disposição (JDBC, RMI, etc.)
  - Não comprometem a estabilidade do servidor em caso de falha (na pior hipótese, um erro poderia derrubar o JVM)

Problemas dos servlets, CGI e APIs

- *Para gerar páginas dinâmicas (99% das aplicações), é preciso embutir o HTML ou XML dentro de instruções de uma linguagem de programação:*

```
out.print("<h1>Servlet</h1>");
for (int num = 1; num <= 5; i++) {
    out.print("<p>Parágrafo " + num + "</p>");
}
out.print("<table><tr><td> ... </tr></table>");
```

- *Maior parte da informação da página é estática, no entanto, precisa ser embutida no código*
- *Afasta o Web designer do processo*
  - *Muito mais complicado programar que usar HTML e JavaScript*
  - *O design de páginas geradas dinamicamente acaba ficando nas mãos do programador (e não do Web designer)*

Solução: scripts de servidor

- *Coloca a linguagem de programação dentro do HTML (e não o contrário)*

```
<h1>Servlet</h1>
 <% for (int num = 1; num <= 5; i++) { %>
   <p>Parágrafo <%= num %></p>
 <%}%>
 <table><tr><td> ... </tr></table>
```

- *Permite o controle da aparência e estrutura da página em softwares de design (DreamWeaver, FrontPage)*
- *Página fica mais legível*
- *Quando houver muita programação, código pode ser escondido em servlets, JavaBeans, componentes (por exemplo: componentes ActiveX, no caso do ASP)*

Controle de sessão

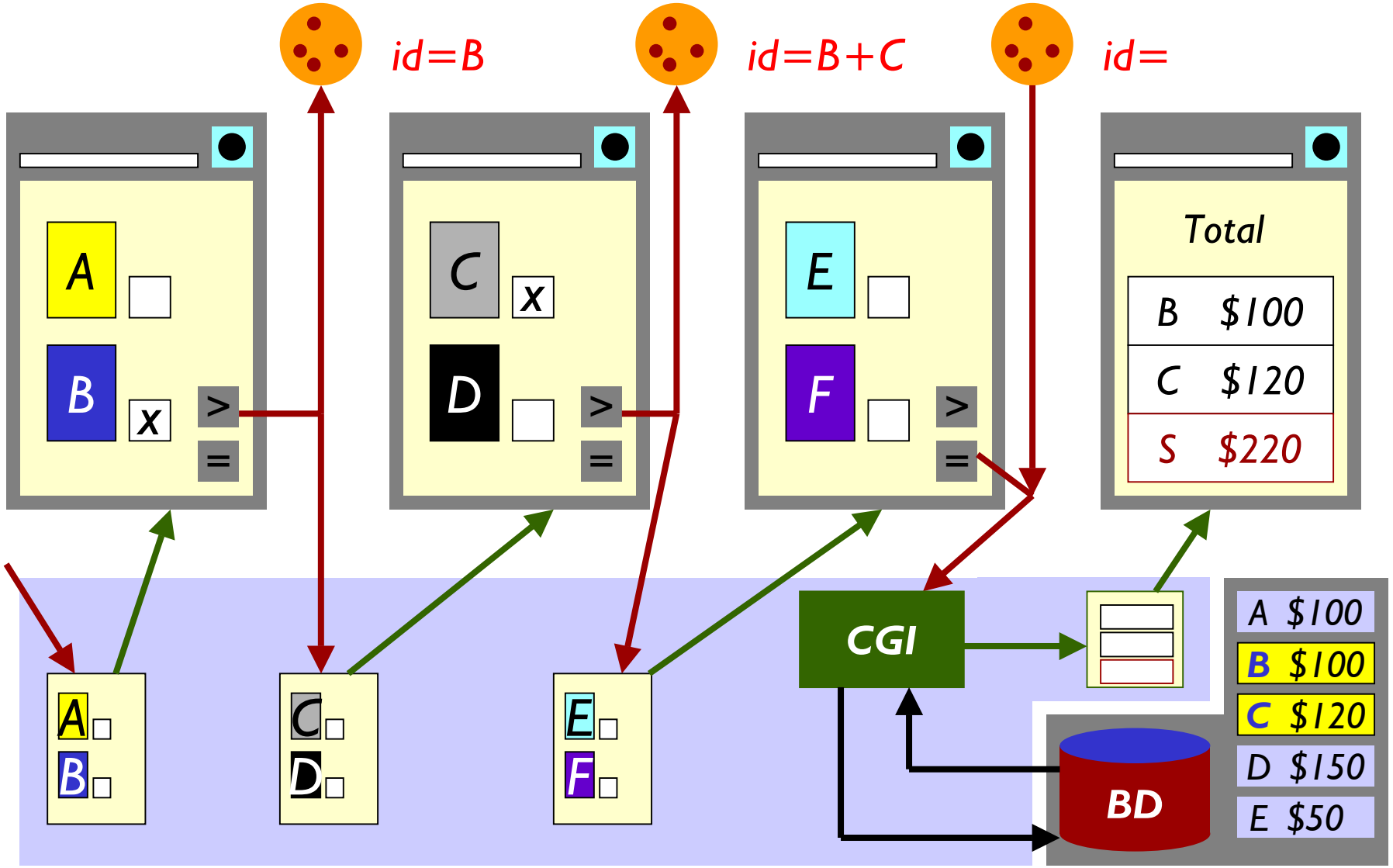
- *HTTP não preserva o estado de uma sessão. É preciso usar mecanismos artificiais com CGI (ou qualquer outra tecnologia Web)*
  - *Seqüência de páginas/aplicações: desvantagens: seqüência não pode ser quebrada; mesmo que página só contenha HTML simples, precisará ser gerada por aplicação*
  - *Inclusão de dados na URL: desvantagens: pouca flexibilidade e exposição de informações*
  - *Cookies (informação armazenada no cliente): desvantagens: espaço e quantidade de dados reduzidos; browser precisa suportar a tecnologia*

Cookies

- *Padrão Internet (RFC) para persistência de informações entre requisições HTTP*
- *Um cookie é uma pequena quantidade de informação que o servidor armazena no cliente*
  - *Par nome=valor. Exemplos: usuario=paulo, num=123*
  - *Escopo no servidor: domínio e caminho da página*
  - *Pode ser seguro*
  - *Escopo no cliente: browser (sessão)*
  - *Duração: uma sessão ou tempo determinado (cookies persistentes)*
- *Cookies são criados através de cabeçalhos HTTP*

```
Content-type: text/html
Content-length: 34432
Set-Cookie: usuario=ax343
Set-Cookie: lastlogin=12%2610%2699
```

Exemplo com cookies: Loja virtual



> *Guarda cookie e chama próxima página*

= *Lê cookie, apaga-o e envia dados para CGI*

Aplicaçoes Web e Java

- Servlets e JavaServer Pages (JSP) são as soluções Java para estender o servidor HTTP
  - Suportam os métodos de requisição padrão HTTP (GET, POST, HEAD,PUT, DELETE, OPTIONS, TRACE)
  - Geram respostas compatíveis com HTTP (códigos de status, cabeçalhos RFC 822)
  - Interagem com Cookies
- Além dessas tarefas básicas, também
  - Suportam filtros, que podem ser chamados em cascata para tratamento de dados durante a transferência
  - Suportam controle de sessão transparentemente através de cookies ou rescrita de URLs (automática)
- É preciso usar um servidor que suporte as especificações de servlets e JSP

Primeiro servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleServlet extends HttpServlet {
    public void doGet (HttpServletRequest  request,
                       HttpServletResponse response)
                          throws ServletException, IOException {
       PrintWriter  out;
       response.setContentType("text/html");
       out = response.getWriter();
       String user = request.getParameter("usuario");
       if (user == null)
           user = "World";

       out.println("<HTML><HEAD><TITLE>");
       out.println("Simple Servlet Output");
       out.println("</TITLE></HEAD><BODY>");
       out.println("<H1>Simple Servlet Output</H1>");
       out.println("<P>Hello, " + user);
       out.println("</BODY></HTML>");
       out.close();
    }
}
```

Primeiro JSP

```
<HTML><HEAD>
<TITLE>Simple Servlet Output</TITLE>
</HEAD><BODY>
<%
      String user =
         request.getParameter("usuario");
      if (user == null)
          user = "World";
%>
<H1>Simple Servlet Output</H1>
<P>Hello, <%= user %>
</BODY></HTML>
```

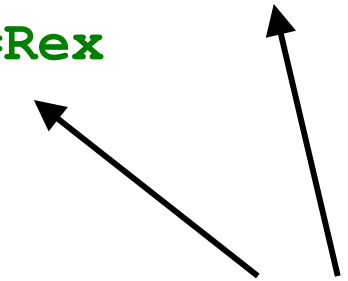
Página recebida no browser

- *Url da requisição*

```
http://servidor/servlet/SimpleServlet?usuario=Rex
http://servidor/hello.jsp?usuario=Rex
```

- *Código fonte visto no cliente*

```
<HTML><HEAD>
<TITLE>
Simple Servlet Output
</TITLE>
</HEAD><BODY>
<H1>Simple Servlet Output</H1>
<P>Hello, Rex
</BODY></HTML>
```



Usando contexto default ROOT no TOMCAT

Um simples JavaBean

```
package beans;

public class HelloBean implements
                        java.io.Serializable {

      private String msg;

      public HelloBean() {
            this.msg = "World";
      }

      public String getMensagem() {
            return msg;
      }

      public void setMensagem(String msg) {
            this.msg = msg;
      }
}
```

Primeiro Bean JSP

- *Página JSP que usa HelloBean.class*

```
<HTML><HEAD>
<jsp:useBean id="hello" class="beans.HelloBean" />
<jsp:setProperty name="hello" property="mensagem"
                 param="usuario" />
<TITLE>
Simple Servlet Output
</TITLE>
</HEAD><BODY>
<H1>Simple Servlet Output</H1>
<P>Hello, <jsp:getProperty name="hello"
                           property="mensagem" />
</BODY></HTML>
```

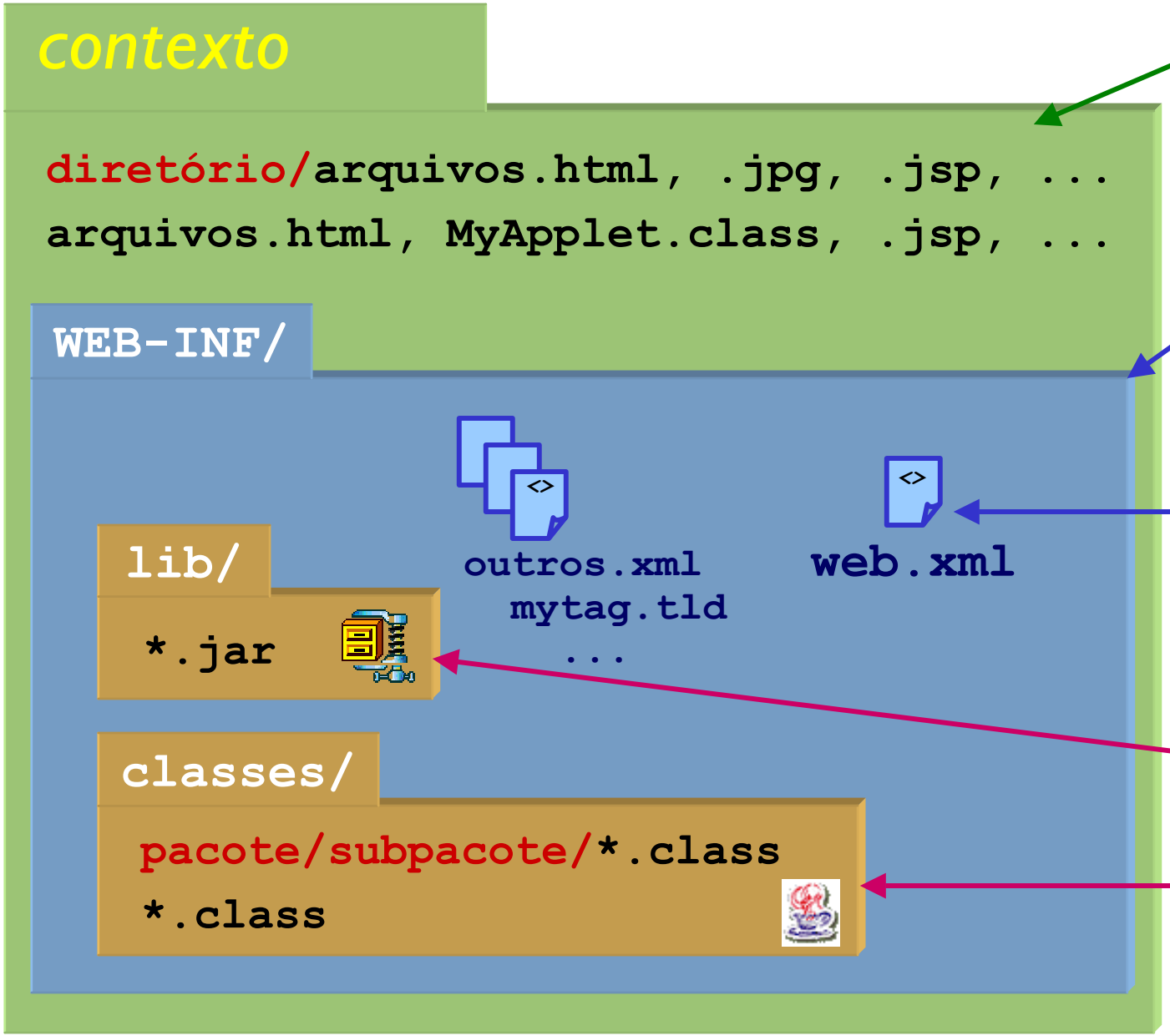
Componentes Web

- São aplicações J2EE
- Rodam em um Web Container que oferece serviços como repasse de requisições, segurança, concorrência (threads), gerência do ciclo de vida
- São compostos de servlets e páginas JSP empacotados em um arquivo WAR (tipo de JAR)
- O WAR é essencial para implantar o cliente Web J2EE, mas opcional em servidor standalone
- Os componentes de um WAR ocupam um contexto que pode ser acessado através de um cliente HTTP (browser)
  - Fisicamente, o contexto representa uma estrutura de diretórios. Logicamente, representa uma aplicação Web.
  - O contexto tem uma estrutura padrão definida em especificação

Aplicações Web (contextos)

- No JBoss, aplicações Web são implantadas copiando contextos compactados em arquivos WAR para a pasta deploy/
  - A estrutura da árvore de diretórios deve ser mantida dentro do arquivo WAR
- Todo diretório de contexto tem uma estrutura definida, que consiste de
  - Área de documentos do contexto (/), acessível externamente
  - Área inacessível (/WEB-INF), que possui pelo menos um arquivo de configuração padrão (web.xml)
  - O WEB-INF pode conter ainda dois diretórios reconhecidos pelo servidor: (1) um diretório que pertence ao CLASSPATH da aplicação (/WEB-INF/classes) e (2) outro onde podem ser colocados JARs para inclusão no CLASSPATH (/WEB-INF/lib)

Estrutura de uma aplicação Web



Arquivos acessíveis ao cliente a partir da raiz do contexto

Área inaccessível ao cliente

Arquivo de configuração (WebApp deployment descriptor)

Bibliotecas

Classpath (Contém Classes, JavaBeans, Servlets)

Componentes de um contexto

- *A raiz define (geralmente) o nome do contexto.*
  - *Na raiz ficam HTMLs, páginas JSP, imagens, applets e outros objetos para download via HTTP*

{Contexto}/WEB-INF/web.xml

  - *Arquivo de configuração da aplicação*
  - *Define parâmetros iniciais, mapeamentos e outras configurações de servlets e JSPs.*

{Contexto}/WEB-INF/classes/

  - *Classpath da aplicação*

{Contexto}/WEB_INF/lib/

  - *Qualquer JAR incluido aqui será carregado como parte do CLASSPATH da aplicação*

Nome do contexto e URL de acesso

- *A não ser que seja configurado externamente, o nome do contexto aparece na URL após o nome/porta do servidor*

```
http://serv:8080/contexto/subdir/pagina.html
http://serv:8080/contexto/servlet/pacote.Servlet
```

- *Para os documentos no servidor (links em páginas HTML e formulários), a raiz de referência é a raiz de documentos do servidor, ou* `DOCUMENT_ROOT`: `http://serv:8080/`
- *Documentos podem ser achados relativos ao* `DOCUMENT_ROOT`

```
/contexto/subdir/pagina.html
/contexto/servlet/pacote.Servlet
```

- *Para a configuração do contexto (web.xml), a raiz de referência é a raiz de documentos do contexto:* `http://serv:8080/contexto/`
- *Componentes são identificados relativos ao contexto*

```
/subdir/pagina.html
/servlet/pacote.Servlet
```

`servlet/` *é mapeamento virtual definido no servidor para servlets em* `WEB-INF/classes`

Tipos e fragmentos de URL

- *URL absoluta: identifica recurso na Internet. Usada no campo de entrada de localidade no browser, em páginas fora do servidor, etc.*

  `http://serv:8080/ctx/servlet/pacote.Servlet/cmd/um`

- *Relativa ao servidor (Request URI): identifica o recurso no servidor. Pode ser usada no código interpretado pelo browser nos atributos HTML que aceitam URLs (para documentos residentes no servidor)*

  `/ctx/servlet/pacote.Servlet/cmd/um`

- *Relativa ao contexto: identifica o recurso dentro do contexto. Pode ser usada no código de servlets e JSP interpretados no servidor e web.xml. Não contém o nome do contexto.*

  `/servlet/pacote.Servlet/cmd/um`

- *Relativa ao componente (extra path information): texto anexado na URL após a identificação do componente ou página*

  `/cmd/um`

Criando um contexto válido

- *Para que uma estrutura de diretórios localizada no webapps/ seja reconhecida como contexto pelo Tomcat, na inicialização, deve haver um arquivo web.xml no diretório WEB-INF do contexto*
  - *O arquivo é um arquivo XML e deve obedecer às regras do XML e do DTD definido pela especificação*
  - *O conteúdo mínimo do arquivo é a declaração do DTD e um elemento raiz <web-app/>*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app/>
```

  - *Se houver qualquer erro no web.xml, a aplicação não será carregada durante a inicialização*

Configuração: exemplo (1/3)

```
<web-app>
  <context-param>
    <param-name>tempdir</param-name>
    <param-value>/tmp</param-value>
  </context-param>

  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>example.MyServlet</servlet-class>
    <init-param>
      <param-name>datafile</param-name>
      <param-value>data/data.txt</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>myJSP</servlet-name>
    <jsp-file>/myjsp.jsp</jsp-file>
    <load-on-startup>2</load-on-startup>
  </servlet>
                    ...
```

Parâmetro que pode ser lido por todos os componentes

Instância de um servlet

Parâmetro que pode ser lido pelo servlet

Ordem para carga prévia do servlet

Instância de servlet de página JSP

Ordem para pré-compilar JSP

Configuração: exemplo (2/3)

```
...
  <servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/myservlet</url-pattern>
  </servlet-mapping>

  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <error-page>
    <error-code>404</error-code>
    <location>/notFound.jsp</location>
  </error-page>
...
```

Servlet examples.myServlet foi mapeado à URL /myservlet

Sessão do usuário expira expira em 30 minutos

Lista de arquivos que serão carregados automaticamente em URLs terminadas em diretório

Redirecionar para esta página em caso de erro 404

Configuração: exemplo (3/3)

Recursos externos acessíveis via JNDI (java:comp/env/jdbc/MeuBanco)

```
...
  <resource-ref>
    <res-ref-name>jdbc/MeuBanco</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>CONTAINER</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
```

Ligação com nome JNDI em outro contexto pode ser feito em arquivo externo (no EAR ou configuração proprietária, ex: jboss-web.xml)

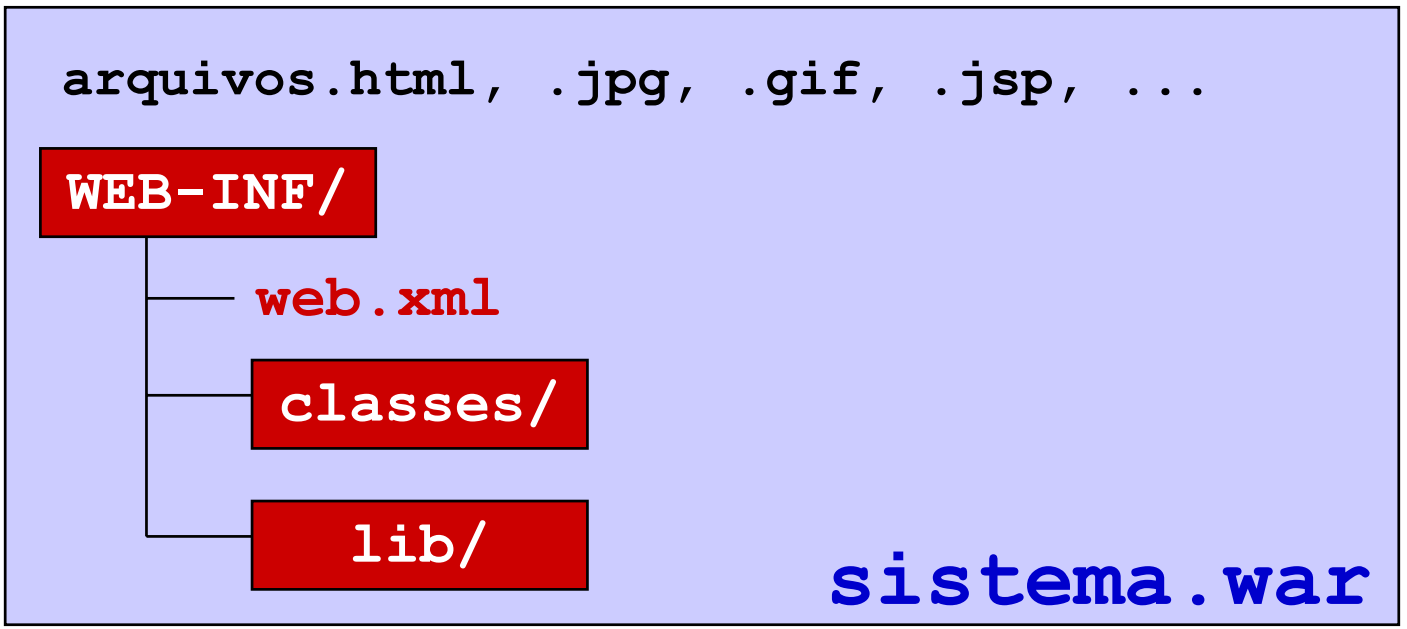
```
  <env-entry>
    <env-entry-name>valor</env-entry-name>
    <env-entry-value>34.45</env-entry-value>
    <env-entry-type>java.lang.Double</env-entry-type>
  </env-entry>

</web-app>
```

Variáveis compartilhadas pelo ambiente

Web Archive

- *Utilizável no Tomcat e também em servidores J2EE*
- *Permite criação de novo contexto automaticamente*
- *Coloque JAR contendo estrutura de um contexto no diretório de deployment (webapps, no Tomcat)*
  - *O JAR deve ter a extensão .WAR*
  - *O JAR deve conter WEB-INF/web.xml válido*

Exemplo - aplicação: `http://servidor/sistema/`



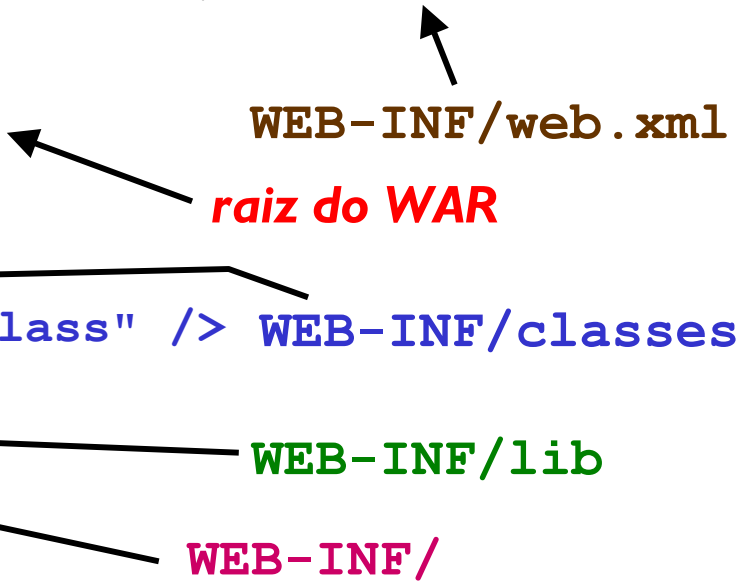
Como criar um WAR

- O mesmo WAR que serve para o Tomcat, serve para o JBoss, Weblogic, WebSphere, etc.
  - Todos aderem à mesma especificação
- Há várias formas de criar um WAR
  - Usando o deploytool do Tomcat.
  - Usando um aplicativo tipo WinZip
  - Usando uma ferramenta JAR:
    jar -cf arquivo.war -C diretorio_base .
  - Usando a ferramenta packager do kit J2EE:
    packager -webArchive <opções>
  - Usando a tarefa <jar> ou <war> no Ant

WAR criado pelo Ant

- *Pode-se criar WARs usando a tarefa <jar> ou <war>*
  - *Com <jar> você precisa explicitamente definir seus diretórios WEB-INF, classes e lib (usando um <zipfileset>, por exemplo) e copiar os arquivo web.xml, suas classes e libs.*
  - *Com <war> você pode usar o atributo webxml, que já coloca o arquivo web.xml no lugar certo, e outros elementos de um war:*

```
<war warfile="bookstore.war" webxml="etc/metainf.xml">
   <fileset dir="web" >
      <include name="*.html />
      <include name="*.jsp" />
   </fileset>
   <classes dir="${build}" >
      <include name="database/*.class" />
   </classes>
   <lib dir="${lib.dir}" />
   <webinf dir="${etc.dir}" />
</war>
```



  - *Veja o manual do Ant para outros exemplos e detalhes*

Configuração externa do WAR (servidores J2EE)

- Configuração externa ao WAR pode ser feita quando WAR é acrescentado em um arquivo EAR e funciona em servidores J2EE (JBoss, por exemplo)
  - EAR é JAR comum com arquivo application.xml no seu META-INF
- O arquivo application.xml do EAR deve conter

```
<application>
  <module>
    <web>
      <web-uri>mywebapp.war</web-uri>
      <context-root>/myroot</context-root>
    </web>
  </module>
</application>
```

- A aplicação agora é acessada via

```
http://servidor/myroot
```

Deployment e execução

- Depende do servidor
  - No JBoss, copie o WAR ou EAR para o diretório deploy do servidor para implantar o serviço. Remova o WAR para tirar o serviço do ar
  - No J2EE Reference Implementation Server, use o deployment wizard
- Para executar, as regras são as mesmas que uma aplicação comum. Acesse o contexto raiz via Web
  - http://servidor/nome-do-contexto/
  - http://servidor/nome-do-contexto/index.jsp
  - http://servidor/nome-do-contexto/subcontexto/aplicacao
  - http://servidor/nome-do-contexto/servlet/pacote.Classe

JBoss

- *Caso seja necessário configurar, no JBoss*
  - *Nomes JNDI para referências declaradas no web.xml*
  - *Nome da raiz do contexto (o EAR sempre tem precedência)*
  - *Configurações de segurança*

  pode-se criar um arquivo jboss-web.xml e empacotá-lo junto com o WAR
- *Para montar, use o DTD jboss-web_3_0.dtd. Exemplo:*

```
<jboss-web>
    <security-domain>java:jaas/mydomain</security-domain>

    <context-root>myroot</context-root>

    <resource-ref>
        <res-ref-name>jdbc/MyBank</res-ref-name>
        <jndi-name>java:/DefaultDS</jndi-name>
    </resource-ref>
</jboss-web>
```

Conclusões

- *Este capítulo apresentou uma visão superficial de aplicações Web*
  - *Detalhes sobre servlets, sessões, contextos e JSP serão vistos nos capítulos seguintes*
  - *Outros detalhes relacionados a aplicações Web não serão abordados: consulte documentação adicional no CD ou material do curso J550 (Servlets e JSP)*
- *Para utilizar componentes Web em aplicações J2EE é essencial saber implantá-las*
  - *Crie WARs e implante-os no JBoss*
  - *Familiarize-se com a sintaxe das URLs e contextos*
  - *Explore os recursos de configuração do web.xml*

Exercícios

- *1. Use os arquivos em cap05/exercicio para montar um contexto cap05 e empacotá-lo em um WAR*
  - *Os arquivos que devem ser copiados para a raiz do WAR (document root) estão na pasta web/*
  - *Os servlets e JavaBeans, que devem ser copiados para WEB-INF/classes após a compilação estão em src/. Após a execução do Ant, serão compilados e colocados em classes/*
  - *O Web Deployment Descriptor (web.xml) está em etc/*

  a) Escreva um script ou alvo do ant para montar o contexto corretamente em um diretório temporário

  b) Crie um WAR contendo os três componentes

  c) Copie o WAR para o diretório deploy/ do JBoss

  d) Acesse as aplicações via browser

  ```
  http://localhost:8080/cap05/hello.jsp
  http://localhost:8080/cap05/servlet/SimpleServlet
  http://localhost:8080/cap05/hellobean.jsp
  ```

Referências

[1] Stephanie Bodoff. Web Clients and Components. J2EE Tutorial, Sun Microsystems, 2002

[2] Fields/Kolb. Web Development with JavaServer Pages, Manning, 2000

[3] Eduardo Pelegri Lopart, Java Server Pages 1.2 Specification, Sun, August 2001. http://java.sun.com. Referência oficial sobre JSP

[4] Danny Coward, Java Servlet Specification 2.3. Sun, August 2001. Referência oficial sobre servlets

[5] Bill Shannon, Java 2 Enterprise Edition Specification v. 1.3, Sun, July 2001 Referência oficial sobre J2EE. Descreve WARs e EARs.

[6] JBoss Group. JBoss User's Manual 2.44. www.jboss.org. Contém referência sobre arquivos de configuração jboss-web.xml

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)

© 1999-2003, Helder da Rocha

Java 2 Enterprise Edition

Helder da Rocha
www.argonavis.com.br

Sobre este módulo

- Neste módulo serão apresentados os fundamentos de servlets
  - Como escrever um servlet
  - Como compilar um servlet
  - Como implantar um servlet no servidor
  - Como executar
- Também serão exploradas as formas de interação do servlet com a requisição e resposta HTTP
  - Como ler parâmetros da entrada
  - Como gerar uma página de resposta
  - Como extrair dados de um formulário HTML

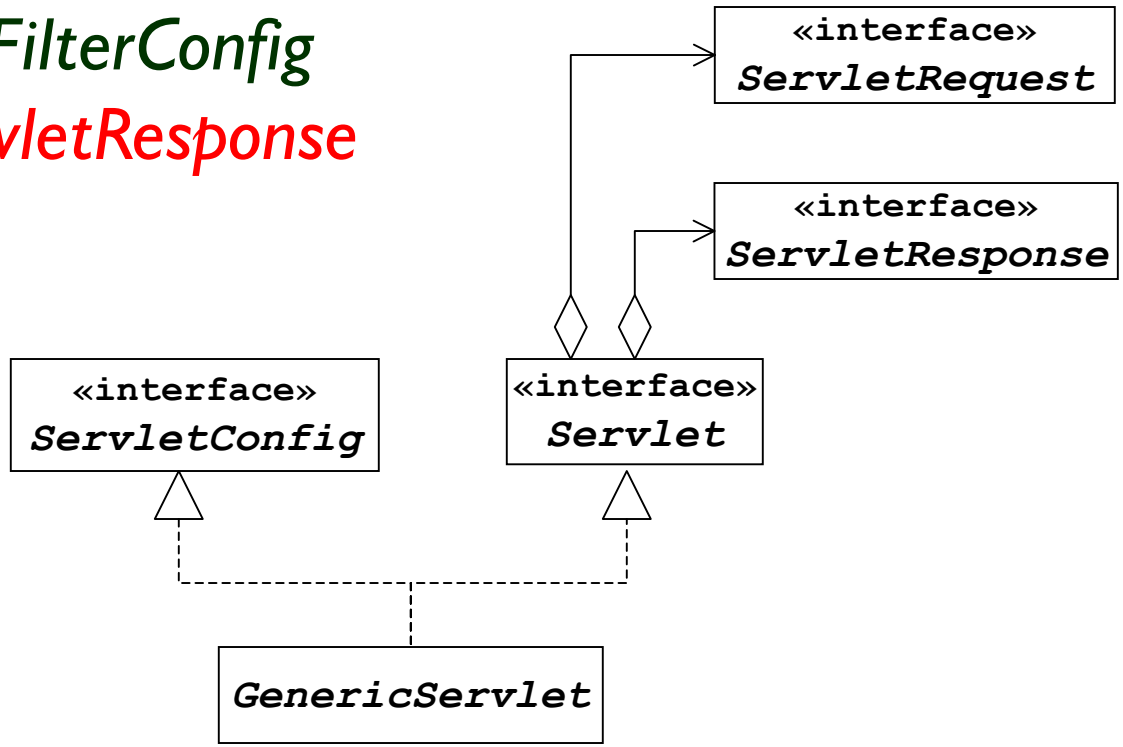
O que são servlets

- Extensão de servidor escrita em Java
  - Servlets são "applets" (pequenas aplicações) de servidor
  - Podem ser usados para estender qualquer tipo de aplicação do modelo requisição-resposta
  - Todo servlet implementa a interface javax.servlet.Servlet (tipicamente estende GenericServlet)
- Servlets HTTP
  - Extensões para servidores Web
  - Estendem javax.servlet.http.HttpServlet
  - Lidam com características típicas do HTTP como métodos GET, POST, Cookies, etc.

API: Servlets

Principais classes e interfaces de javax.servlet

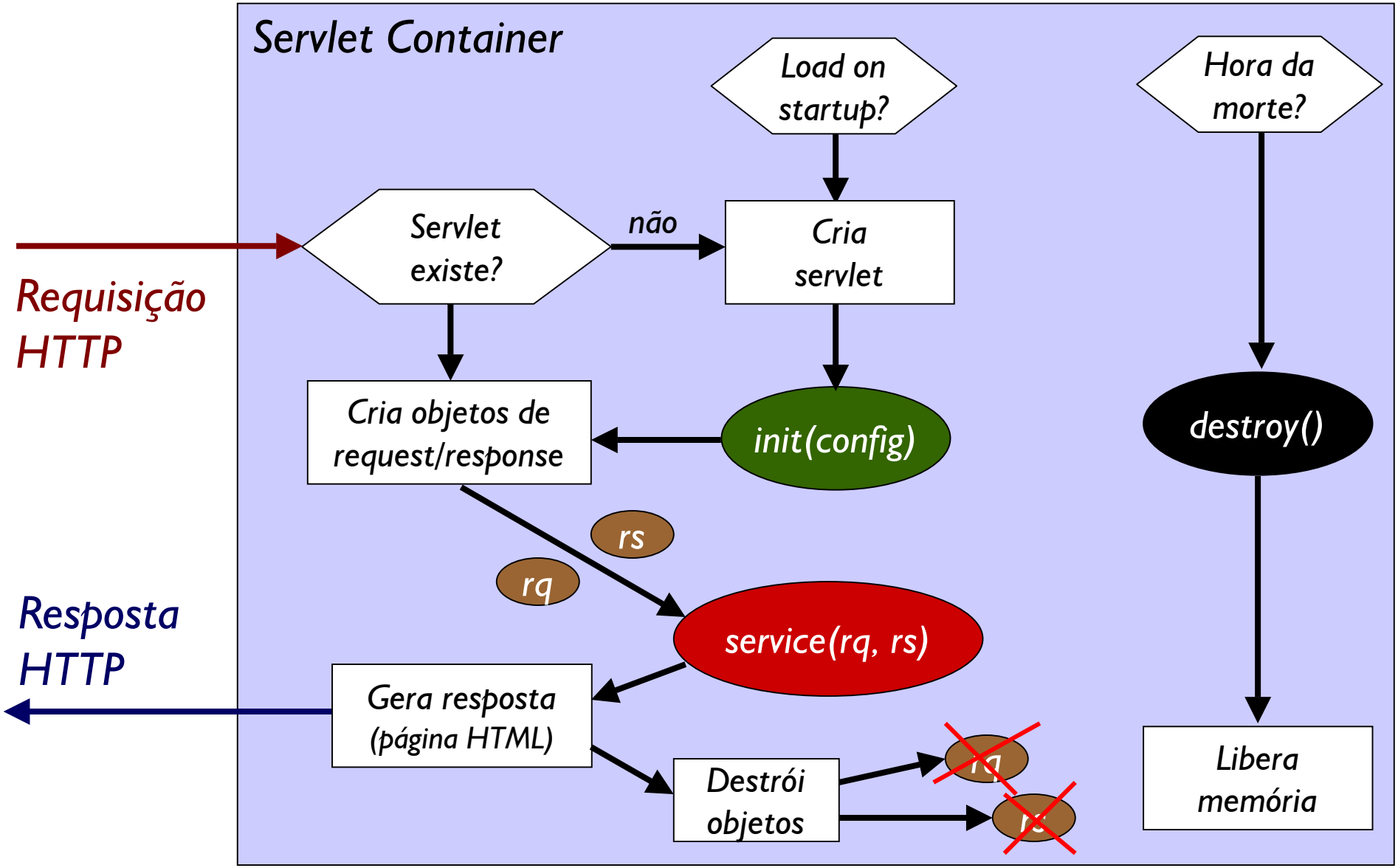
- *Interfaces*
  - *Servlet, ServletConfig, ServletContext*
  - *Filter, FilterChain, FilterConfig*
  - *ServletRequest, ServletResponse*
  - *SingleThreadModel*
  - *RequestDispatcher*
- *Classes abstratas*
  - *GenericServlet*
- *Classes concretas*
  - *ServletException*
  - *UnavailableException*
  - *ServletInputStream e ServletOutputStream*



Ciclo de vida

- O ciclo de vida de um servlet é controlado pelo container
- Quando o servidor recebe uma requisição, ela é repassada para o container que a delega a um servlet. O container
  1. Carrega a classe na memória
  2. Cria uma instância da classe do servlet
  3. Inicializa a instância chamando o método init()
- Depois aue o servlet foi inicializado, cada requisição é executada em um método service()
  - O container cria um objeto de requisição (ServletRequest) e de resposta (ServletResponse) e depois chama service() passando os objetos como parâmetros
  - Quando a resposta é enviada, os objetos são destruídos
- Quando o container decidir remover o servlet da memória, ele o finaliza chamando destroy()

Ciclo de vida



Métodos de serviço

- São os métodos que implementam operações de resposta executadas quando o cliente envia uma requisição
- Todos os métodos de serviço recebem dois parâmetros: um objeto ServletRequest e outro ServletResponse
- Tarefas usuais de um método de serviço
  - extrair informações da requisição
  - acessar recursos externos
  - preencher a resposta (no caso de HTTP isto consiste de preencher os cabeçalhos de resposta, obter um stream de resposta e escrever os dados no stream)

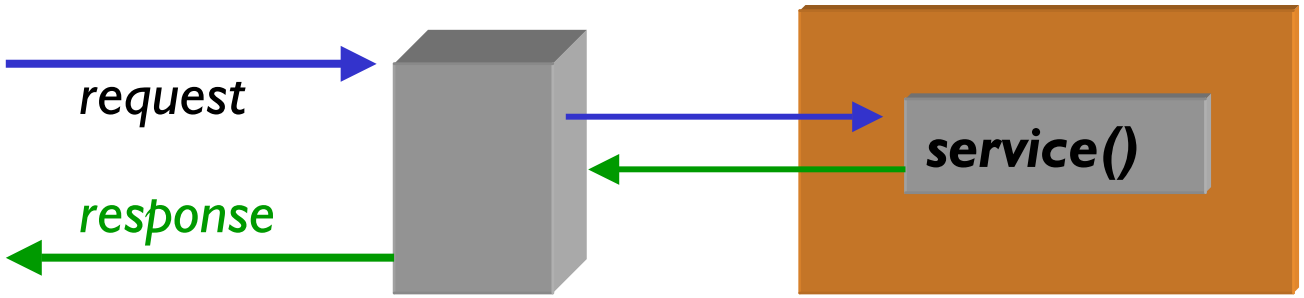
Métodos de serviço (2)

- *O método de serviço de um servlet genérico é o método abstrato service()*

```
public void service(ServletRequest, ServletResponse)
```

definido em javax.servlet.Servlet.

- *Sempre que um servidor repassar uma requisição a um servlet, ele chamará o método service(request, response).*

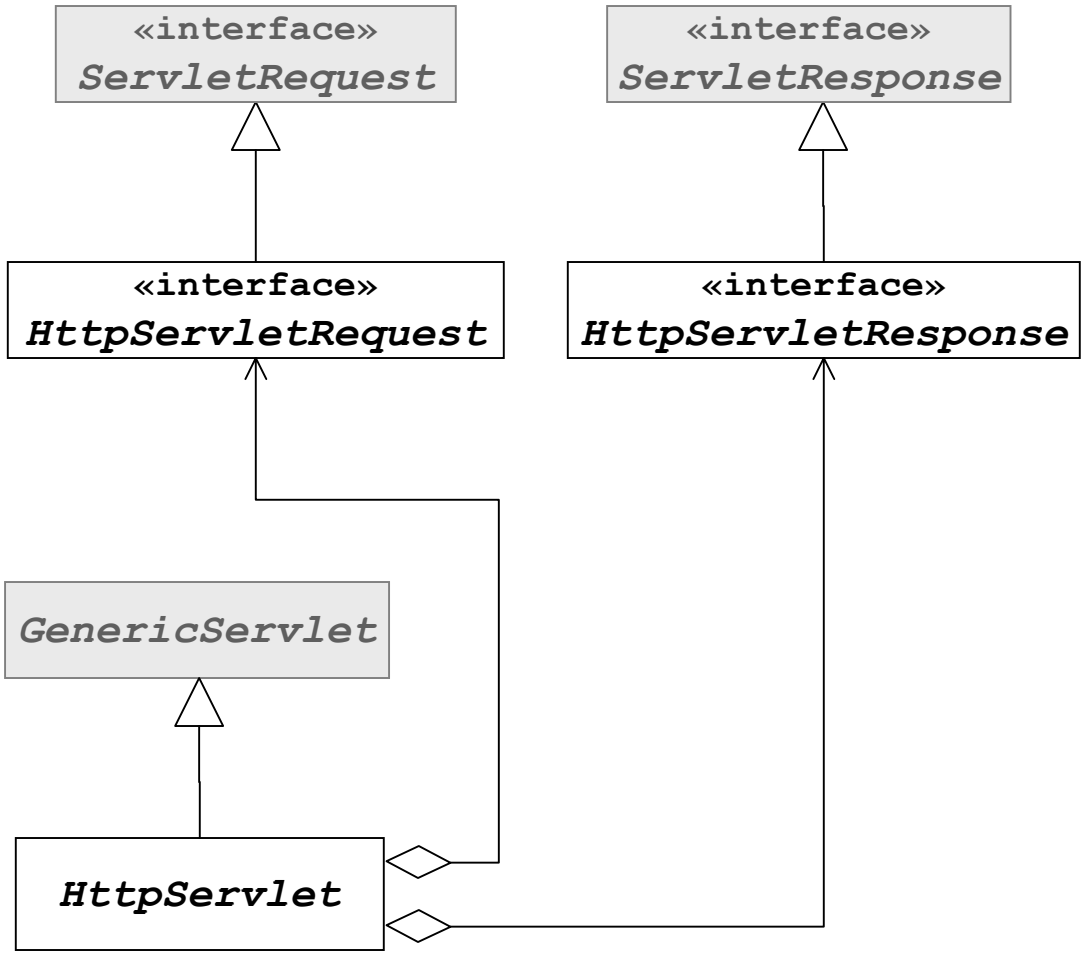


- *Um servlet genérico deverá sobrepor este método e utilizar os objetos ServletRequest e ServletResponse recebidos para ler os dados da requisição e compor os dados da resposta, respectivamente*

API: Servlets HTTP

Classes e interfaces mais importantes do pacote javax.servlet.http

- *Interfaces*
  - *HttpServletRequest*
  - *HttpServletResponse*
  - *HttpSession*
- *Classes abstratas*
  - *HttpServlet*
- *Classes concretas*
  - *Cookie*



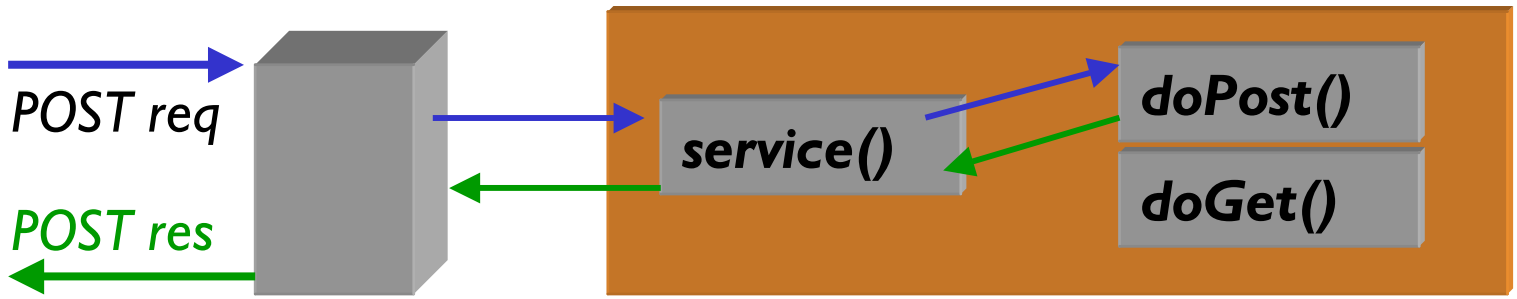
Como escrever um servlet HTTP

- *Para escrever um servlet HTTP, deve-se estender HttpServlet e implementar um ou mais de seus métodos de serviço, tipicamente: doPost() e/ou doGet()*

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ServletWeb extends HttpServlet {
   public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
                           throws IOException {

      PrintWriter out = response.getWriter();
      response.setContentType("text/html");
      out.println("<h1>Hello, World!</h1>");
      out.close();
   }
}
```

Métodos de serviço HTTP

- *A classe HttpServlet redireciona os pedidos encaminhados para service() para métodos que refletem os métodos HTTP (GET, POST, etc.):*
  - *public void doGet(HttpServletRequest, HttpServletResponse)*
  - *public void doPost(HttpServletRequest, HttpServletResponse)*
  - *… **



- *Um servlet HTTP genérico deverá estender HTTPServlet e implementar pelo menos um dos métodos doGet() ou doPost()*

** doDelete(), doTrace(), doPut(), doOptions() - Método HEAD é implementado em doGet()*

Inicialização

- *A inicialização de um GenericServlet, como o HttpServlet, deve ser feita com o método init()*
- *Todos os métodos de config estão no servlet, pois GenericServlet implementa ServletConfig*

```
public void init() throws ServletException {
    String dirImagens =
                getInitParameter("imagens");
    if (dirImagens == null) {
        throw new UnavailableException
                        ("Configuração incorreta!");
    }
}
```

A requisição HTTP

- *Uma requisição HTTP feita pelo browser tipicamente contém vários cabeçalhos RFC822**

```
GET /docs/index.html HTTP/1.0
Connection: Keep-Alive
Host: localhost:8080
User-Agent: Mozilla 6.0 [en] (Windows 95; I)
Accept: image/gif, image/x-bitmap, image/jpg, image/png, */*
Accept-Charset: iso-8859-1, *
Cookies: jsessionid=G3472TS9382903
```

- *Os métodos de HttpServletRequest permitem extrair informações de qualquer um deles*
  - *Pode-se também identificar o método e URL*

* *Especificação de cabeçalho para e-mail*

Obtenção de dados de requisições

- Alguns métodos de HttpServletRequest
  - Enumeration getHeaderNames() - obtém nomes dos cabeçalhos
  - String getHeader("nome") - obtém primeiro valor do cabeçalho
  - Enumeration getHeaders("nome") - todos os valores do cabeçalho
  - String getParameter(param) - obtém parâmetro HTTP
  - String[] getParameterValues(param) - obtém parâmetros repetidos
  - Enumeration getParameterNames() - obtém nomes dos parâmetros
  - Cookie[] getCookies() - recebe cookies do cliente
  - HttpSession getSession() - retorna a sessão
  - setAttribute("nome", obj) - define um atributo obj chamado "nome".
  - Object getAttribute("nome") - recupera atributo chamado nome
  - String getRemoteUser() - obtém usuário remoto (se autenticado, caso contrátio devolve null)

A resposta HTTP

- *Uma resposta HTTP é enviada pelo servidor ao browser e contém informações sobre os dados anexados*

```
HTTP/1.0 200 OK
Content-type: text/html
Date: Mon, 7 Apr 2003 04:33:59 GMT-03
Server: Apache Tomcat/4.0.4 (HTTP/1.1 Connector)
Connection: close
Set-Cookie: jsessionid=G3472TS9382903

<HTML>
   <h1>Hello World!</h1>
</HTML>
```

- *Os métodos de HttpServletResponse permitem construir um cabeçalho*

Preenchimento de uma resposta

- Alguns métodos de HttpServletResponse
  - addHeader(String nome, String valor) - adiciona cabeçalho HTTP
  - setContentType(tipo MIME) - define o tipo MIME que será usado para gerar a saída (text/html, image/gif, etc.)
  - sendRedirect(String location) - envia informação de redirecionamento para o cliente (Location: url)
  - Writer getWriter() - obtém um Writer para gerar a saída. Ideal para saída de texto.
  - OutputStream getOutputStream() - obtém um OutputStream. Ideal para gerar formatos diferentes de texto (imagens, etc.)
  - addCookie(Cookie c) - adiciona um novo cookie
  - encodeURL(String url) - envia como anexo da URL a informação de identificador de sessão (sessionid)
  - reset() - limpa toda a saída inclusive os cabeçalhos
  - resetBuffer() - limpa toda a saída, exceto cabeçalhos

Como implementar doGet() e doPost()

- Use *doGet()* para receber requisições GET
  - Links clicados ou URL digitadas diretamente
  - Alguns formulários que usam GET
- Use *doPost()* para receber dados de formulários
- Se quiser usar ambos os métodos, não sobrepoha service() mas implemente tanto doGet() como doPost()

```
public class ServletWeb extends HttpServlet {
   public void doGet (HttpServletRequest request,
                      HttpServletResponse response) {
       processar(request, response);
   }
   public void doPost (HttpServletRequest request,
                       HttpServletResponse response) {
       processar(request, response);
   }
   public void processar(HttpServletRequest request,
                         HttpServletResponse response) {
       ...
   }
}
```

Parâmetros da requisição

- Parâmetros são pares nome=valor que são enviados pelo cliente concatenados em strings separados por &:

```
nome=Jo%E3o+Grand%E3o&id=agente007&acesso=3
```

- Parâmetros podem ser passados na requisição de duas formas
  - Se o método for GET, os parâmetros são passados em uma única linha no query string, que estende a URL após um "?"

```
GET /servlet/Teste?id=agente007&acesso=3 HTTP/1.0
```

  - Se o método for POST, os parâmetros são passados como um stream no corpo na mensagem (o cabeçalho Content-length, presente em requisições POST informa o tamanho

```
POST /servlet/Teste HTTP/1.0
Content-length: 21
Content-type: x-www-form-urlencoded

id=agente007&acesso=3
```

Como ler parâmetros da requisição

- *Caracteres reservados e maiores que ASCII-7bit são codificados em URLs:*
  - *Ex:* `ã = %E3`
- *Formulários HTML codificam o texto ao enviar os dados automaticamente*
- *Seja o método POST ou GET, os valores dos parâmetros podem ser recuperados pelo método getParameter() de ServletRequest, que recebe seu nome*

```
String parametro = request.getParameter("nome");
```

- *Parâmetros de mesmo nome podem ser repetidos. Neste caso getParameter() retornará apenas a primeira ocorrência. Para obter todas use String[] getParameterValues()*

```
String[] params = request.getParameterValues("nome");
```

Como gerar uma resposta

- *Para gerar uma resposta, primeiro é necessário obter, do objeto HttpServletResponse, um fluxo de saída, que pode ser de caracteres (Writer) ou de bytes (OutputStream)*

```
Writer out = response.getWriter();    // ou
OutputStream out = response.getOutputStream();
```

- *Apenas um deve ser usado. Os objetos correspondem ao mesmo stream de dados*
- *Deve-se também definir o tipo de dados a ser gerado. Isto é importante para que o cabeçalho Content-type seja gerado corretamente e o browser saiba exibir as informações*

```
response.setContentType("text/html");
```

- *Depois, pode-se gerar os dados, imprimindo-os no objeto de saída (out) obtido anteriormente*

```
out.println("<h1>Hello</h1>");
```

Compilação e Implantação de servlets

- Para compilar, use qualquer distribuição da API
  - O servlet.jar distribuído pelo Tomcat em common/lib/
  - O j2ee.jar distribuído no pacote J2EE da Sun (em lib/)
  - O javax.servlet.jar do JBoss (server/default/lib/)
- Inclua o JAR no seu CLASSPATH ao compilar

```
> javac -classpath ../servlet.jar;. MeuServlet.java
```

- Para implantar, copie as classes compiladas para um contexto existente no servidor
  - Jakarta-Tomcat (webapps/ROOT/WEB-INF/classes)
  - JBoss: (server/default/deploy/)

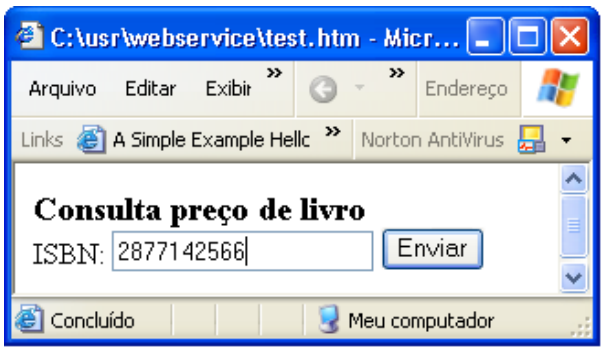
Usando o Ant

- Pode-se usar o Ant para fazer a compilação e deployment de uma aplicação Web
  - Defina um <classpath> adicional nas tarefas <javac> que inclua o caminho do servlet.jar
  - Use <copy> para copiar os arquivos para os contextos corretos
  - Use <property environment="env" /> e as propriedades `env.TOMCAT_HOME` ou `env.CATALINA_HOME` para ler as variáveis de ambiente do sistema e tornar seu build file independente da localização do servidor
- O Ant também pode ser usado para gerar o JAR que encapsula uma aplicação Web (WAR) usando a tarefa <war>

Execução

- Use
  - http://localhost:8080/contexto/servlet/nome.do.Servlet
- Para passar parâmetros
  - Passe-os via URL, acrescentando um ? seguido dos pares nome=valor (separados por "&"):

    `http://localhost:8080/servlet/Servlet?isbn=123456`
  - Ou escreva um formulário HTML




```
<FORM ACTION="/servlet/Servlet"
      METHOD="POST">
 <H3>Consulta preço de livro</H3>
 <P>ISBN: <INPUT TYPE="text" NAME="isbn">
 <INPUT TYPE="Submit" VALUE="Enviar">
</FORM>
```

Exercícios

Para os exercícios abaixo, use um contexto cap06.

- 1. Crie um servlet (exercicio.ParameterList) que imprima, em uma tabela, todos os nomes de parâmetros enviados e seus valores
  - O servlet deve suportar tanto GET como POST
  - Use request.getParameterNames() e getParameterValues(String)
  - Use o formulário allForm.html fornecido para testá-lo
- 2. Crie um servlet (exercicio.HeaderList) que imprima, em uma tabela, todos os nomes de cabeçalhos HTTP da requisição e seus valores
  - O servlet deve suportar tanto GET como POST
  - Use request.getHeaderNames() e getHeaders(String)

Instâncias de servlets

- *Uma instância de um servlet pode ser configurada no web.xml através do elemento <servlet>*

```
<servlet>
   <servlet-name>myServlet</servlet-name>
   <servlet-class>exemplo.pacote.MyServlet</servlet-class>
   <!-- elementos de configuração opcionais aqui -->
</servlet>
```

  - *<servlet-name> e <servlet-class> são obrigatórios*
- *É uma boa prática escolher nomes de servlets seguindo as convenções Java*
  - *Use caixa mista, colocando em maiúsculas cada palavra, mas comece com letra minúscula. Ex: banco, pontoDeServico*
- *Pode-se criar múltiplas instâncias da mesma classe definindo blocos servlet com <servlet-name> diferentes*
  - *Não terão muita utilidade a não ser que tenham também configuração diferente e mapeamentos diferentes*

Servlet alias (mapeamento) no web.xml

- *É uma boa prática definir aliases para os servlets*
  - *Nomes grandes são difíceis de digitar e lembrar*
  - *Expõem detalhes sobre a implementação das aplicações*
- *Para definir um mapeamento de servlet é necessário usar <servlet> e <servlet-mapping>*
- *<servlet-mapping> associa o nome do servlet a um padrão de URL relativo ao contexto. A URL pode ser*
  - *Um caminho relativo ao contexto iniciando por /*
  - *Uma extensão de arquivo, expresso da forma *.extensao*

```
<servlet>
    <servlet-name>umServlet</servlet-name>
    <servlet-class>pacote.subp.MeuServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>umServlet</servlet-name>
    <url-pattern>/programa</url-pattern>
</servlet-mapping>
```

Sintaxe de mapeamentos

- *Mapeamento exato*
  - *Não aceita /nome/ ou /nome/x na requisição*

```
<url-pattern>/nome</url-pattern>
<url-pattern>/nome/subnome</url-pattern>
```

- *Mapeamento para servlet default*
  - *Servlet é chamado se nenhum dos outros mapeamentos existentes combinar com a requisição*

```
<url-pattern>/</url-pattern>
```

- *Mapeamento de caminho*
  - *Aceita texto adicional (path info) após nome do servlet na requisição*

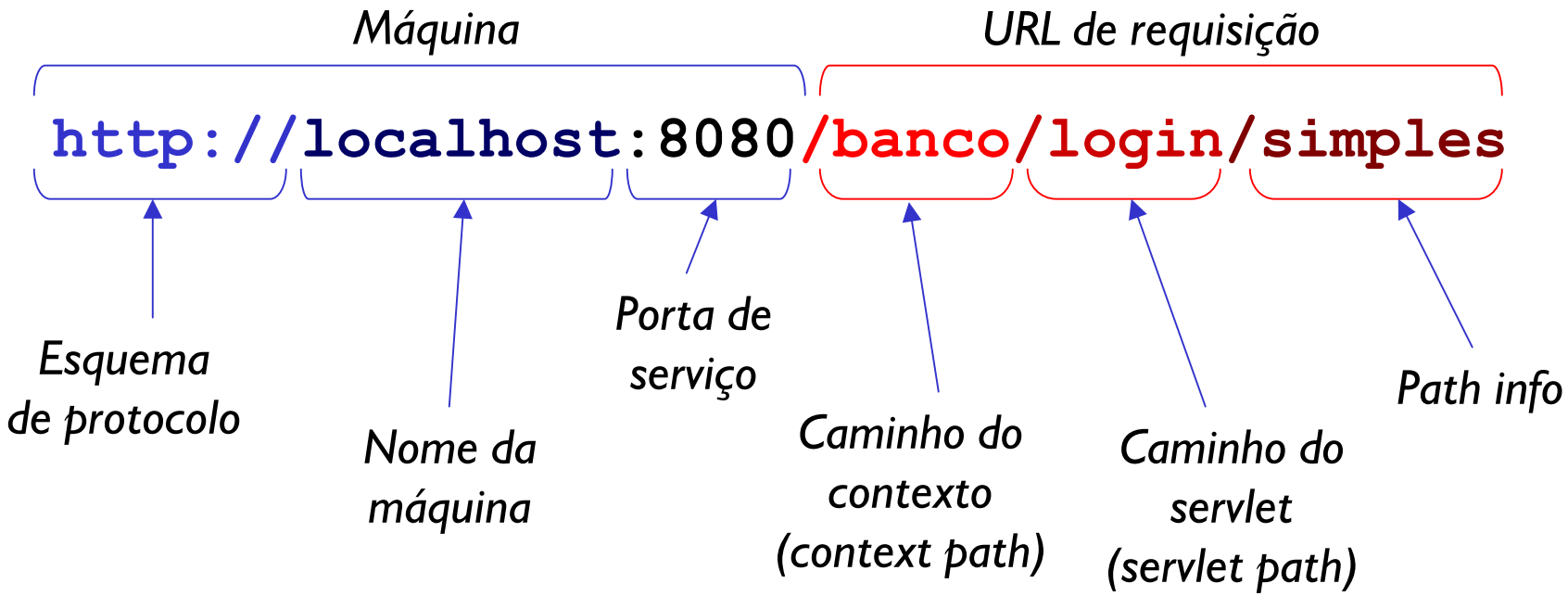
```
<url-pattern>/nome/*</url-pattern>
<url-pattern>/nome/subnome/*</url-pattern>
```

- *Mapeamento de extensão*
  - *Arquivos com a extensão serão redirecionados ao servlet*

```
<url-pattern>*.ext</url-pattern>
```

Anatomia de uma URL

- *Diferentes partes de uma URL usada na requisição podem ser extraídas usando métodos de HttpServletRequest*
  - `getContextPath()`: */banco*, *na URL abaixo*
  - `getServletPath()`: */login*, *na URL abaixo*
  - `getPathInfo()`: */simples*, *na URL abaixo*



ServletConfig

- A interface ServletConfig serve para que um servlet possa ter acesso a informações de configuração definidas no web.xml
- Todo servlet implementa ServletConfig e, portanto, tem acesso aos seus métodos
- Principais métodos de interesse
  - String getInitParameter(String nome): lê um parâmetro de inicialização <init-param> do web.xml
  - Enumeration getInitParameterNames(): obtém os nomes de todos os parâmetros de inicialização disponíveis
- Os métodos de ServletConfig devem ser chamados no método init(), do servlet

Definição de parâmetros de inicialização

- Parâmetros de inicialização podem ser definidos para cada instância de um servlet usando o elemento <init-param> dentro de <servlet>
  - Devem aparecer depois de <servlet-name> e <servlet-class> (lembre-se que a ordem foi definida no DTD)
  - Requer dois sub-elementos que definem o nome do atributo e o seu valor

```
<servlet>
    <servlet-name>umServlet</servlet-name>
    <servlet-class>pacote.subp.MeuServlet</servlet-class>
    <init-param>
      <param-name>dir-imagens</param-name>
      <param-value>c:/imagens</param-value>
    </init-param>
    <init-param> ... </init-param>
  </servlet>
```

Leitura de parâmetros de inicialização

- *Parâmetros de inicialização podem ser lidos no método init() e guardados em variáveis de instância para posterior uso dos métodos de serviço*

```
private java.io.File dirImagens = null;

public void init() throws ServletException {
    String dirImagensStr =
              getInitParameter("dir-imagens");
    if (dirImagens == null) {
        throw new UnavailableException
                    ("Configuração incorreta!");
    }
    dirImagens = new File(dirImagensStr);
    if (!dirImagens.exists()) {
       throw new UnavailableException
                    ("Diretorio de imagens nao existe!");
    }
}
```

ServletContext

- A interface ServletContext encapsula informações sobre o contexto ou aplicação
- Cada servlet possui um método getServletContext() que devolve o contexto atual
  - A partir de uma referência ao contexto atual pode-se interagir com o contexto e compartilhar informações entre servlets
- Principais métodos de interesse de ServletContext
  - String getInitParameter(String): lê parâmetros de inicialização do contexto (não confunda com o método similar de ServletConfig!)
  - Enumeration getInitParameterNames(): lê lista de parâmetros
  - InputStream getResourceAsStream(): lê recurso localizado dentro do contexto como um InputStream
  - setAttribute(String nome, Object): grava um atributo no contexto
  - Object getAttribute(String nome): lê um atributo do contexto
  - log(String mensagem): escreve mensagem no log do contexto

Inicialização de contexto

- No web.xml, <context-param> vem antes de qualquer definição de servlet

```
<context-param>
    <param-name>tempdir</param-name>
    <param-value>/tmp</param-value>
</context-param>
```

- No servlet, é preciso primeiro obter uma instância de ServletContext antes de ler o parâmetro

```
ServletContext ctx = getServletContext();
String tempDir = ctx.getInitParameter("tempdir");
if (tempDir == null) {
   throw new UnavailableException("Configuração errada");
}
```

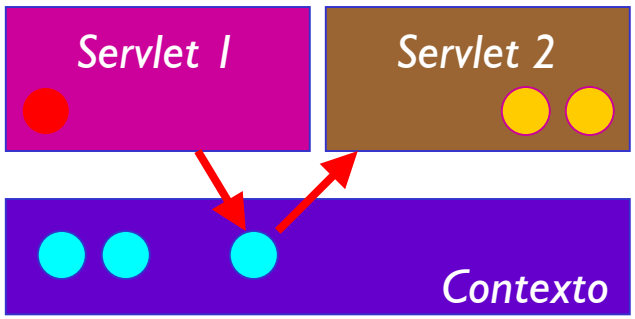
Carregamento de arquivos no contexto

- O método getResourceAsStream() permite que se localize e se carregue qualquer arquivo no contexto sem que seja necessário saber seu caminho completo
  - Isto é importante pois contextos podem ser usados em diferentes servidores e armazenados em arquivos WAR
- Exemplo

```
ServletContext ctx = getServletContext();
String arquivo = "/WEB-INF/usuarios.xml";
InputStream stream = ctx.getResourceAsStream(arquivo);
InputStreamReader reader =
                    new InputStreamReader(stream);
BufferedReader in = new BufferedReader(reader);
String linha = "";
while ( (linha = in.readLine()) != null) {
   // Faz alguma coisa com linha de texto lida
}
```

Gravação de atributos no contexto

- *Servlets podem compartilhar objetos pelo contexto usando*
  - `setAttribute("nome", objeto);`
  - `Object getAttribute("nome");`



- *Exemplo de uso*

Servlet 1

```
String[] vetor = {"um", "dois", "tres"};
ServletContext ctx = getServletContext();
ctx.setAttribute("dados", vetor);
```

Servlet 2

```
ServletContext ctx = getServletContext();
String[] dados = (String[])ctx.getAttribute("dados");
```

- *Outros métodos*
  - `removeAttribute(String nome)` - *remove um atributo*
  - `Enumeration getAttributeNames()` - *lê nomes de atributos*

Escopo e threads

- Geralmente, só há uma instância de um servlet rodando para vários clientes
  - Atributos de instância são compartilhados!
- Se não desejar compartilhar dados entre clientes, use sempre objetos thread-safe
  - Atributos guardados no request
  - Variáveis locais
- Quaisquer outros atributos, como atributos de sessão, atributos de instância e de contexto são compartilhados entre requisições
  - Caso deseje compartilhá-los, use synchronized nos blocos de código onde seus valores são alterados.

Repasse de requisição

- Objetos *RequestDispatcher* servem para repassar requisições para outra página ou servlet. Seus dois principais métodos são

```
include(request, response)
```

```
forward(request, response)
```

- Esses métodos não podem definir cabeçalhos
  - *forward()* repassa a requisição para um recurso
  - *include()* inclui a saída e processamento de um recurso no servlet
- Para obter um RequestDispatcher use o ServletRequest

```
RequestDispatcher dispatcher =
  request.getRequestDispatcher("url");
```

- Para repassar a requisição para outra máquina use

```
dispatcher.forward(request, response);
```

- No repasse de requisição, o controle não volta para o browser.
  - Todos os parâmetros e atributos da requisição são preservados

Redirecionamento x Repasse

- Pode-se enviar um cabeçalho de redirecionamento para o browser usando

```
response.sendRedirect("url");
```

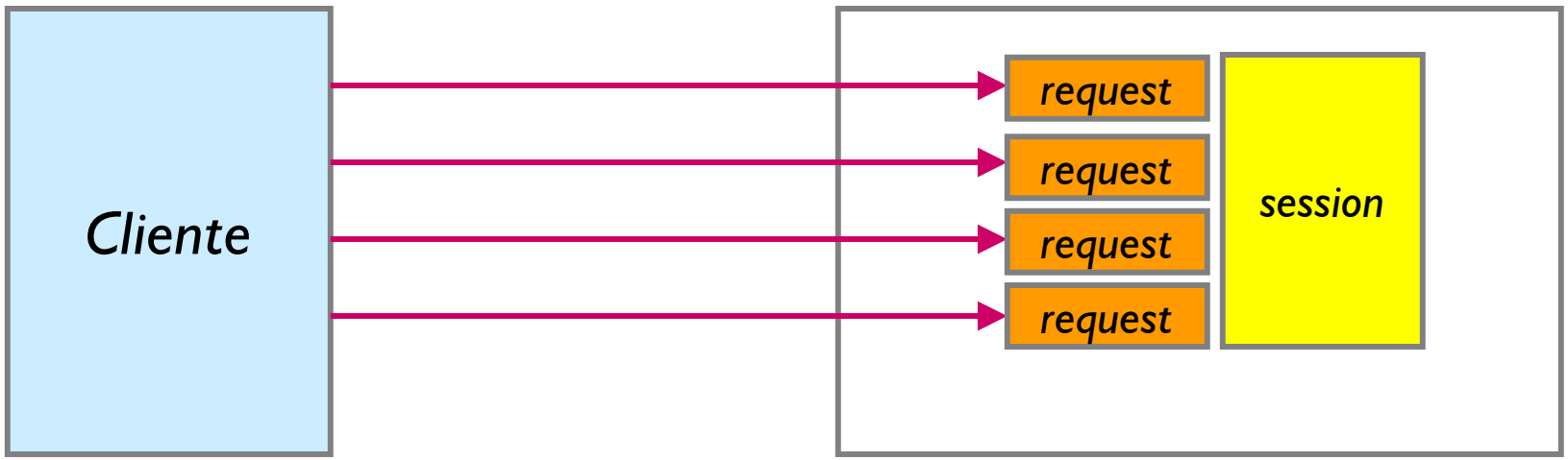
- Isto é o mesmo que fazer

```
response.setHeader("Location", "url");
```

- Location é um cabeçalho HTTP que instrui o browser para redirecionar para outro lugar
- Sempre que o controle volta ao browser, a primeira requisição terminou e outra foi iniciada
  - Os objetos HttpServletResponse e HttpServletRequest e todos seus atributos e parâmetros foram destruídos
- Com repasse de requisições, usando RequestDispatcher, o controle não volta ao browser mas continua em outro servlet (com forward())ou no mesmo servlet (com include())

Sessões

- *Como o HTTP não mantém estado de sessão, são as aplicações Web que precisam cuidar de mantê-lo quando necessário*
- *Sessões representam um cliente*
  - *A sessão é única para cada cliente e persiste através de várias requisições*



Sessões

- Sessões são representados por objetos *HttpSession* e são obtidas a partir de uma requisição
- Dois métodos podem ser usados

```
HttpSession session = request.getSession(false);
```

  - Se a sessão não existir, retorna null, caso contrário retorna sessão.

```
HttpSession session = request.getSession();
```

  - Retorna a sessão ou cria uma nova. Mesmo que *getSession(true)*
- Para saber se uma sessão é nova, use o método *isNew()*

```
if (session.isNew()) {
  myObject = new BusinessObject();
} else {
  myObject = (BusinessObject) session.getAttribute("obj");
}
```

- *getSession()* deve ser chamado antes de *getOutputStream()**
  - Sessões podem ser implementadas com cookies, e cookies são definidos no cabeçalho HTTP (que é montado antes do texto)

*ou qualquer método que obtenha o stream de saída, como getWriter()

Compartilhamento de objetos na sessão

- *Dois métodos*
  - `setAttribute("nome", objeto);`
  - `Object getAttribute("nome");`

  permitem o compartilhamento de objetos na sessão. Ex:

  Requisição 1

```
String[] vetor = {"um", "dois", "tres"};
HttpSession session = request.getSession();
session.setAttribute("dados", vetor);
```

  Requisição 2

```
HttpSession session = request.getSession();
String[] dados = (String[])session.getAttribute("dados");
```

- *Como a sessão pode persistir além do tempo de uma requisição, é possível que a persistência de alguns objetos não sejam desejáveis*
  - *Use* `removeAttribute("nome")` *para remover objetos da sessão*

Sessão à prova de clientes

- A sessão é implementada com cookies se o cliente suportá-los
  - Caso o cliente não suporte cookies, o servidor precisa usar outro meio de manter a sessão
- Solução: sempre que uma página contiver uma URL para outra página da aplicação, a URL deve estar dentro do método encodeURL() de HttpServletResponse

```
out.print("<a href='" +
                response.encodeURL("caixa.jsp") + "'>");
```

- Se cliente suportar cookies, URL passa inalterada (o identificador da sessão será guardado em um cookie)
- Se cliente não suportar cookies, o identificador será passado como parâmetro da requisição

  ex: http://localhost:8080/servlet/Teste;jsessionid=A424JX08S99

Escopo de objetos em servlets

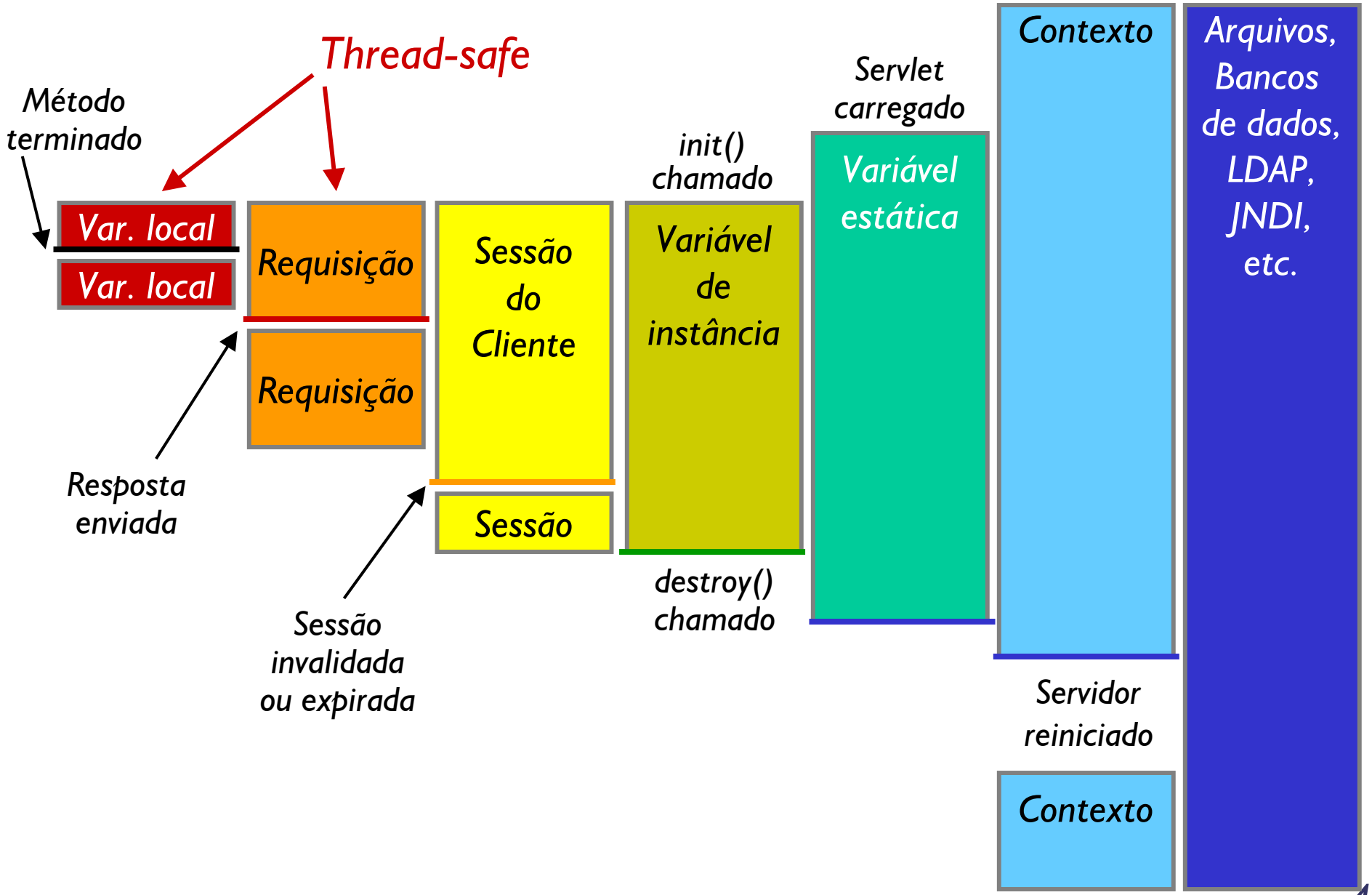
- Servlets podem compartilhar informações de várias maneiras
  - Usando meios persistentes (bancos de dados, arquivos, etc)
  - Usando objetos na memória por escopo (requisição, sessão, contexto)
  - Usando variáveis estáticas ou de instância
- Servlets oferecem três níveis diferentes de persistência na memória (ordem decrescente de duração)
  - *Contexto da aplicação*: vale enquanto aplicação estiver na memória (javax.servlet.*ServletContext*)
  - *Sessão*: dura uma sessão do cliente (javax.servlet.http.*HttpSession*)
  - *Requisição*: dura uma requisição (javax.servlet.*ServletRequest*)
- Para gravar dados em um objeto de persistência na memória

```
objeto.setAttribute("nome", dados);
```

- Para recuperar ou remover os dados

```
Object dados = objeto.getAttribute("nome");
objeto.removeAttribute("nome");
```

Escopo de objetos em servlets: resumo



Lidando com recursos compartilhados

- Há vários cenários de acesso concorrente
  - Componentes compartilhando sessão ou contexto
  - Threads acessando variáveis compartilhadas
- Servlets são automaticamente multithreaded
  - O container cria um thread na instância para cada requisição
  - É preciso sincronizar blocos críticos para evitar problemas decorrentes do acesso paralelo
- Exemplo: protegendo definição de atributo de contexto:

```
synchronized(this) {
    context.setAttribute("nome", objeto);
}
```

- Para situações onde multithreading é inaceitável, servlet deve implementar a interface SingleThreadModel (só um thread estará presente no método service() ao mesmo tempo)
  - Evite isto a todo custo: muito ineficiente!

Acesso a bancos de dados

- *Servlets são aplicações Java e, como qualquer outra aplicação Java, podem usar JDBC e integrar-se com um banco de dados relacional*
- *Pode-se usar java.sql.DriverManager e obter a conexão da forma tradicional*

```
Class.forName("nome.do.Driver");
Connection con =
     DriverManager.getConnection("url", "nm", "ps");
```

- *Pode-se obter as conexões de um pool de conexões através de javax.sql.DataSource via JNDI (use esta solução em servidores J2EE!)*

```
DataSource ds = (DataSource)ctx.lookup("jbdc/Banco");
Connection con = ds.getConnection();
```

Recursos em servidores J2EE

- Servlets rodando em servidores compatíveis J2EE podem acessar recursos através de JNDI (dominio java:comp/env)
  - Variáveis (environment entries)
  - Referências para componentes EJB
  - Referências para fábricas de recursos (conexões de banco de dados, URLs, serviço de e-mail, JMS, conectores EIS via JCA)
  - Serviços
- Para usar esses recursos
  - Servlet deve estar empacotado em um WAR
  - Nome das variáveis e referências devem ser declarados no web.xml
  - Servlet deve usar como contexto inicial o domínio java:comp/env
- Elementos (filhos de <web-app>) usados no web.xml
  - <env-entry>
  - <ejb-ref>
  - <resource-ref>

Environment Entries

- *Alternativa global (para o WAR) aos <init-param>*
  - *São acessíveis dentro de qualquer servlet ou JSP da aplicação WAR*
  - *Não são visíveis por outras aplicações do servidor (não é um nome JNDI global - está abaixo de java:comp/env - é local à aplicação)*
  - *Acessíveis via ferramentas de deployment (podem ser redefinidas)*
- *Exemplo de uso dentro do <web-app>*

```
<env-entry>
  <env-entry-name>cores/fundo</env-entry-name>
  <env-entry-value>rgb(255, 255, 200)</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

  - *Tipos de dados legais são String e wrappers (Double, Integer, etc.)*
- *Uso dentro do servlet*

```
Context initCtx = new InitialContext();
String fgColor = (String)
                 initCtx.lookup("java:comp/env/cores/fundo");
```

Componentes EJB

- *Servlets e JSPs podem se comunicar com EJBs da aplicação declarando uma referência associada ao bean chamado*
  - *A referência deve informar o tipo do bean (Session, Entity ou MessageDriven e suas interfaces remota e home.*

```
<ejb-ref>
   <description>Cruise ship cabin</description>
   <ejb-ref-name>ejb/CabinHome</ejb-ref-name>
   <ejb-ref-type>Entity</ejb-ref-type>
   <home>com.titan.cabin.CabinHome</home>
   <remote>com.titan.cabin.Cabin</remote>
</ejb-ref>
```

- *Componentes EJB são retornados como objetos CORBA que precisam ser reduzidos através da função narrow.*

```
InitialContext initCtx = new InitialContext();
Object ref = initCtx.lookup("java:comp/env/ejb/CabinHome");
CabinHome home = (CabinHome)
          PortableRemoteObject.narrow(ref, CabinHome.class);
```

Banco de dados

- *Fábricas de objetos são acessíveis via <resource-ref>. A mais comum é a fábrica de conexões de banco de dados*

```
<resource-ref>
  <description>Cloudscape database</description>
  <res-ref-name>jdbc/BankDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>SERVLET</res-auth>
</resource-ref>
```

  - *<res-auth> informa quem é responsável pela autenticação*

- *Através da DataSource, obtém-se uma conexão.*

```
InitialContext initCtx = new InitialContext();
DataSource ds = (DataSource)
                initCtx.lookup("java:comp/env/jdbc/BankDB");
Connection con1 = ds.getConnection();// res-auth: CONTAINER
Connection con2 =
   source.getConnection("user","pass"); // res-auth: SERVLET
```

Conclusões

- Neste módulo abordamos os assuntos mais importantes sobre servlets, contextos e sessões
  - Para mais detalhes sobre esses assuntos consulte as referências no CD ou os slides do curso J550
- O próximo módulo abordará JSP, cuja API básica é a mesma usada nos servlets
  - Familiarizar-se com a construção e uso da API de servlets ajuda na implementação de páginas JSP mais eficientes
  - Mesmo com aplicações consistindo basicamente de páginas JSP, servlets têm papel importante na separação de responsabilidades (arquiteturas MVC, FrontController, etc.)

Exercícios

- 3. Crie aliases para os servlets criados nos exercícios anteriores 1 e 2:
  - /parameters para exercicio.ParameterList
  - /headers para exercicio.HeaderList
- 4. Parâmetros de inicialização
  - a) Guarde, como parâmetros de inicialização de ParameterList, duas cores: corDeFundo e corDoTexto que receba nomes de cor
  - b) Gere a página usando os parâmetros para colori-la
  - c) Use um parâmetro de contexto para gravar a fonte usada em todos os servlets do contexto.
- 5. Uso de Session
  - a) Crie um arquivo index.html contendo um link para /headers e um formulário, que receba um 'nome' e envie-o para /parameters usando POST
  - b) Em /parameters, grave o 'nome' na sessão
  - c) Em /headers, leia o nome da sessão e imprima-o

Fontes

[1] *Jason Hunter, William Crawford. Java Servlet Programming, 2nd edition, O'Reilly and Associates, 2001*

[2] *J2EE Tutorial*

[3] *Marty Hall, Core Servlets and JSP, Prentice-Hall, 2000*

[4] *Jim Farley et. al. Java Enterprise in a NutShell, 2nd. edition, O'Reilly and Associates, April 2002*

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)

© 1999-2003, Helder da Rocha

Java 2 Enterprise Edition

JavaServer Pages

Helder da Rocha
www.argonavis.com.br

Objetivos

- Este módulo apresenta JavaServer Pages
  - Sintaxe dos marcadores JSP e objetos
  - Funcionamento
  - Como implantar
- Tudo o que vale para servlets continua valendo para JavaServer Pages
  - Um JSP é um servlet durante a execução
  - Escrever código em JSP é como escrever código dentro do doPost() ou doGet() de um servlet com os objetos response, request, out, session e outros já definidos
  - Um JSP, depois de carregado, é tão veloz quando um servlet
  - É mais fácil escrever e implantar, mas é mais difícil depurar

Problemas de servlets

- *Servlets forçam o programador a embutir código HTML dentro de código Java*
  - *Desvantagem se a maior parte do que tem que ser gerado é texto ou código HTML estático*
  - *Mistura as coisas: programador tem que ser bom Web Designer e se virar sem ferramentas de Web Design*

```
Date hoje = new Date();
out.println("<body>");
out.println("<p>A data de hoje é "+hoje+".</p>");
out.println("<body>");
```

HojeServlet.java

- *Uma solução inteligente é escrever um arquivo de template*

```
<body>
<p>A data de hoje é <!--#data#-->.</p>
<body>
```

template.html

Usando templates em servlets

- *Tendo-se um template, é preciso criar um mecanismo eficiente para processá-los*
  - *No exemplo mostrado, pode-se ler o arquivo seqüencialmente , jogando tudo na saída até achar a seqüência "<!--#"*
  - *Depois, interpretar o "comando", gera-se o código relacionado com ele e prossegue-se na leitura e impressão do resto do documento*
  - *Há várias formas de implementar. O código abaixo usa o pacote javax.util.regex para localizar os comandos e fazer a substituição*

```
Date hoje = new Date();
String pagina = abreHTML("template.html");
Pattern p = Pattern.compile("<!--#data#-->");
Matcher m = p.matcher(pagina);
m.replaceAll(hoje);
out.println(m.toString());
```

HojeServlet.java

- *Com o tempo, define-se um vocabulário e procura-se fazer o processador de templates cada vez mais reutilizável*

O que são JavaServer Pages (JSP)

- JSP é uma tecnologia padrão, baseada em templates para servlets. O mecanismo que a traduz é embutido no servidor.
- Há várias outras alternativas populares
  - Apache Cocoon XSP: baseado em XML (xml.apache.org/cocoon)
  - Jakarta Velocity (jakarta.apache.org/velocity)
  - WebMacro (www.webmacro.org)
- Solução do problema anterior usando templates JSP

```
<body>
<p>A data de hoje é <%=new Date() %>.</p>
<body>
```

hoje.jsp

- Em um servidor que suporta JSP, processamento de JSP passa por uma camada adicional onde a página é transformada (compilada) em um servlet
- Acesso via URL usa como localizador a própria página

Exemplos de JSP

- *A forma mais simples de criar documentos JSP, é*
  1. *Mudar a extensão de um arquivo HTML para .jsp*
  2. *Colocar o documento em um servidor que suporte JSP*
- *Fazendo isto, a página será transformada em um servlet*
  - *A compilação é feita no primeiro acesso*
  - *Nos acessos subseqüentes, a requisição é redirecionada ao servlet que foi gerado a partir da página*
- *Transformado em um JSP, um arquivo HTML pode conter blocos de código (scriptlets):* `<% ... %>` *e expressões* `<%= ... %>` *que são os elementos mais frequentemente usados*

```
<p>Texto repetido:
<% for (int i = 0; i < 10; i++) { %>
  <p>Esta é a linha <%=i %>
<% }%>
```

Exemplo de JSP

```
<%@ page import="java.util.*" %>
<%@ page import="j2eetut.webhello.MyLocales" %>
<%@ page contentType="text/html; charset=iso-8859-9" %>
<html><head><title>Localized Dates</title></head><body bgcolor="white">
<a href="index.jsp">Home</a>
<h1>Dates</h1>
<jsp:useBean id="locales" scope="application"
                          class="j2eetut.webhello.MyLocales"/>
<form name="localeForm" action="locale.jsp" method="post">
<b>Locale:</b><select name=locale>
<%
   Iterator i = locales.getLocaleNames().iterator();
   String selectedLocale = request.getParameter("locale");
   while (i.hasNext()) {
      String locale = (String)i.next();
      if (selectedLocale != null && selectedLocale.equals(locale) ) {
         out.print("<option selected>" + locale + "</option>");
      } else { %>
         <option><%=locale %></option>
<%    }
   } %>
</select><input type="submit" name="Submit" value="Get Date">
</form>
<p><jsp:include page="date.jsp" flush="true" />
</body></html>
```

diretivas

bean

scriptlet

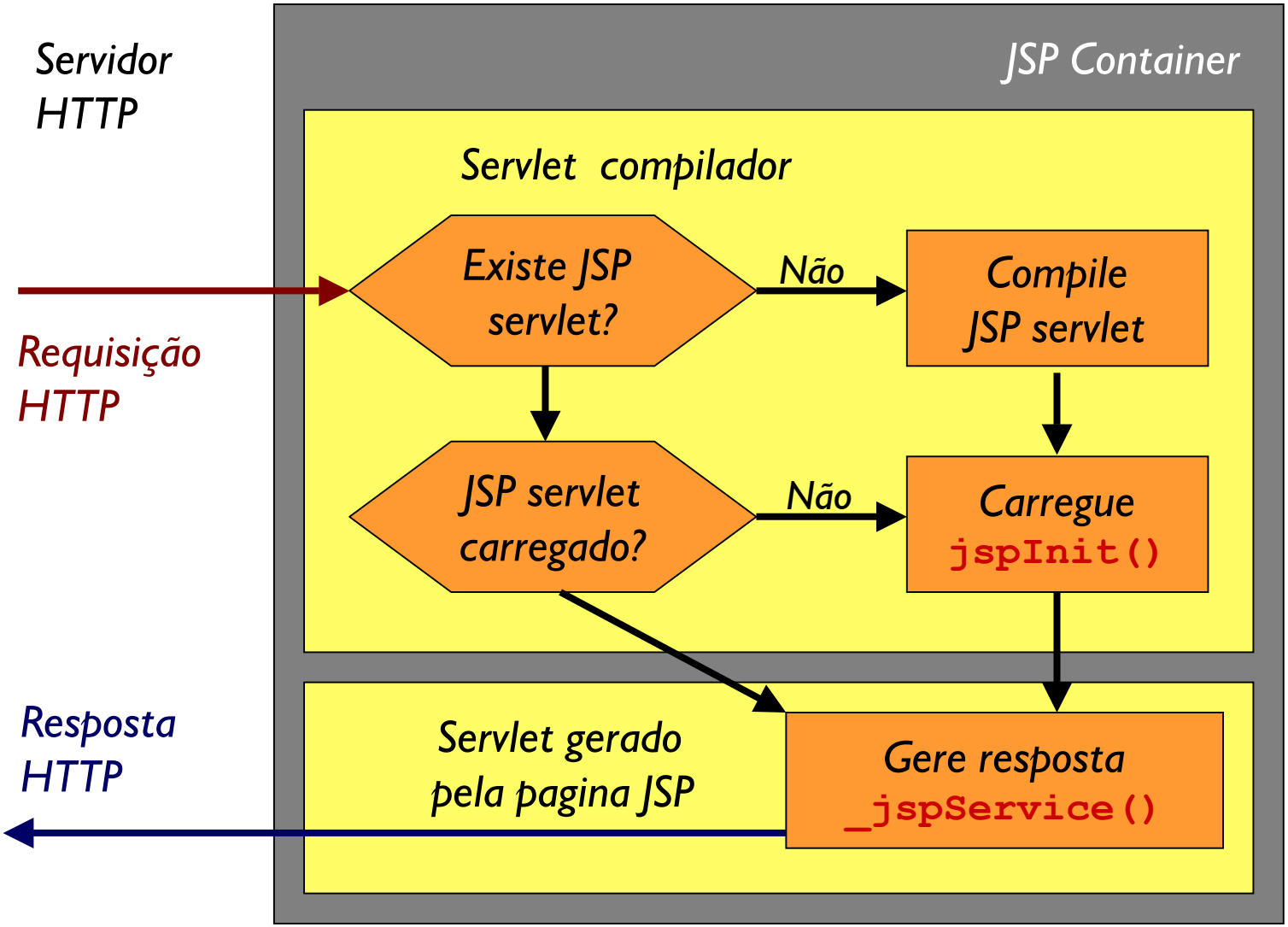
expressão

ação

Ciclo de vida

- Quando uma requisição é mapeada a uma página JSP, o container
  - Verifica se o servlet correspondente à página é mais antigo que a página (ou se não existe)
  - Se o servlet não existe ou é mais antigo, a página JSP será compilada para gerar novo servlet, em seguida, a requisição é repassada ao servlet
  - Se o servlet está atualizado, a requisição é redirecionada para ele
- Deste ponto em diante, o comportamento equivale ao ciclo de vida do servlet, mas os métodos são diferentes
  - Se o servlet ainda não estiver na memória, ele é instanciado, carregado e seu método `jspInit()` é chamado
  - Para cada requisição, seu método `_jspService(req, res)` é chamado. Ele é resultado da compilação do corpo da página JSP
  - No fim da vida, o método `jspDestroy()` é chamado

Como funciona JSP



Sintaxe dos elementos JSP

- *Podem ser usados em documentos de texto (geralmente HTML ou XML)*
- *Todos são interpretados no servidor (jamais chegam ao browser)*
  - *diretivas:* `<%@ ... %>`
  - *declarações:* `<%! ... %>`
  - *expressões:* `<%= ... %>`
  - *scriptlets:* `<% ... %>`
  - *comentários:* `<%-- ... --%>`
  - *ações:* `<jsp:ação ... />`
  - *custom tags:* `<prefixo:elemento ... />`

(a) diretivas

- *Contém informações necessárias ao processamento da classe do servlet que gera a página JSP*
- *Sintaxe :*

```
<%@ diretiva atrib1 atrib2 ... %>
```

- *Principais diretivas:*
  - **page**: *atributos relacionados à página*
  - **include**: *inclui outros arquivos na página*
  - **taglib**: *declara biblioteca de custom tags usada no documento*
- *Exemplos*

```
<%@ page   import="java.net.*, java.io.*"
           session="false"
           errorPage="/erro.jsp" %>
<%@ include file="navbar.jsp" %>
```

(a) diretiva page

- *Atributos de* `<%@page ... %>`

`info="Texto informativo"`	*default: nenhum*
`language="java"`	*(default)*
`contentType="text/html; charset=ISO-8859-1"`	*(default)*
`extends="acme.FonteJsp"`	*default: nenhum*
`import="java.io.*, java.net.*"`	*default: java.lang*
`session="true"`	*(default)*
`buffer="8kb"`	*(default)*
`autoFlush="true"`	*(default)*
`isThreadSafe="true"`	*(default)*
`errorPage="/erros/404.jsp"`	*default: nenhum*
`isErrorPage= "false"`	*(default)*

(b) declarações

- *Dão acesso ao corpo da classe do servlet. Permitem a declaração de variáveis e métodos em uma página*
- *Úteis para declarar:*
  - *Variáveis e métodos de instância (pertencentes ao servlet)*
  - *variáveis e métodos estáticos (pertencentes à classe do servlet)*
  - *Classes internas (estáticas e de instância), blocos static, etc.*
- *Sintaxe*

```
<%! declaração %>
```

- *Exemplos*

```
<%! public final static String[] meses =
      {"jan", "fev", "mar", "abr", "mai", "jun"};
%>
<%! public static String getMes() {
       Calendar cal = new GregorianCalendar();
       return meses[cal.get(Calendar.MONTH)];
    }
%>
```

(b) declarações (métodos especiais)

- *jspInit()* e *jspDestroy()* permitem maior controle sobre o ciclo de vida do servlet
  - Ambos são opcionais
  - Úteis para inicializar conexões, obter recursos via JNDI, ler parâmetros de inicialização do web.xml, etc.
- Inicialização da página (chamado *uma* vez, antes da primeira requisição, após o instanciamento do servlet)

```
<%!
   public void jspInit() { ... }
%>
```

- Destruição da página (ocorre quando o servlet deixa a memória)

```
<%! public void jspDestroy() { ... } %>
```

(c) expressões e (d) scriptlets

- ***Expressões***: *Quando processadas, retornam um valor que é inserido na página no lugar da expressão*
- *Sintaxe:*

```
<%= expressão %>
```

- *Equivale a* `out.print(expressão)`, *portanto, não pode terminar em ponto-e-vírgula*
  - *Todos os valores resultantes das expressões são convertidos em String antes de serem redirecionados à saída padrão*
- ***Scriptlets***: *Blocos de código que são executados sempre que uma página JSP é processada*
- *Correspondem a inserção de seqüências de instruções no método* `_jspService()` *do servlet gerado*
- *Sintaxe:*

```
<% instruções Java; %>
```

(e) comentários

- *Comentários HTML* <!-- --> *não servem para comentar JSP*

```
<!-- Texto ignorado pelo browser mas não pelo
servidor. Tags são processados -->
```

- *Comentários JSP: podem ser usados para comentar blocos JSP*

```
<%--    Texto, código Java, <HTML> ou tags
<%JSP%> ignorados pelo servidor --%>
```

- *Pode-se também usar comentários Java quando dentro de scriptlets, expressões ou declarações:*

```
<%  código JSP ... /* texto ou comandos Java
ignorados pelo servidor */ ... mais código %>
```

(f) ações padronizadas

- *Sintaxe:*

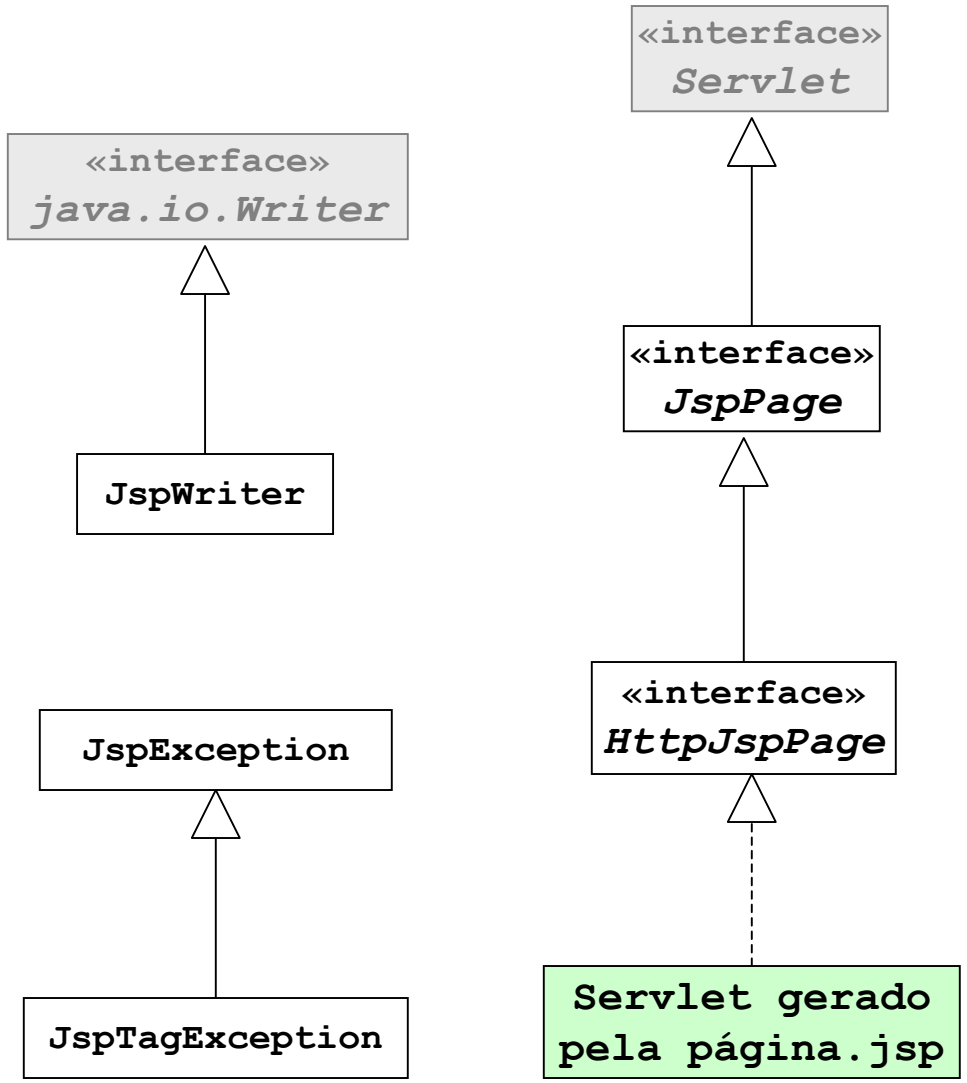
```
<jsp:nome_ação atrib1 atrib2 ... >
  <jsp:param name="xxx" value="yyy"/>
   ...
</jsp:nome_ação>
```

- *Permitem realizar operações (e meta-operações) externas ao servlet (tempo de execução)*
  - *Concatenação de várias páginas em uma única resposta*
    <jsp:forward> e **<jsp:include>**
  - *Inclusão de JavaBeans*
    <jsp:useBean>, **<jsp:setProperty>** e **<jsp:getProperty>**
  - *Geração de código HTML para Applets*
    <jsp:plugin>

API: Classes de suporte a JSP

Pacote *javax.servlet.jsp*

- *Interfaces*
  - *JspPage*
  - *HttpJspPage*
- *Classes abstratas:*
  - *JspEngineInfo*
  - *JspFactory*
  - *JspWriter*
  - *PageContext*
- *Classes concretas:*
  - *JspException*
  - *JspTagException*



Objetos implícitos JSP

- São variáveis locais previamente inicializadas
- Disponíveis nos blocos <% . . . %> (scriptlets) de qualquer página (exceto session e exception que dependem de @page para serem ativados/desativados)

- Objetos do servlet
  - page
  - config
- Entrada e saída
  - request
  - response
  - out
- Objetos contextuais
  - session
  - application
  - pageContext
- Controle de exceções
  - exception

(a) page

- Referência para o servlet gerado pela página
  - Equivale a "this" no servlet
- Pode ser usada para chamar qualquer método ou variável do servlet ou superclasses
  - Tem acesso aos métodos da interface *javax.servlet.jsp.JspPage* (ou *HttpJspPage*)
  - Pode ter acesso a mais variáveis e métodos se estender alguma classe usando a diretiva @page extends:

```
<%@ page extends="outra.Classe" %>
```

- Exemplo:

```
<% HttpSession sessionCopy =
                    page.getSession() %>
```

(b) config

- *Referência para os parâmetros de inicialização do servlet (se existirem) através de objeto ServletConfig*
- *Equivale a page.getServletConfig()*
- *Exemplo:*

```
<% String user = config.getInitParameter("nome");
   String pass = config.getInitParameter("pass"); %>
```

- *Parâmetros de inicialização são fornecidos na instalação do servlet no servidor, através de <init-param> de <servlet> em web.xml. É preciso declarar a página no web.xml*

```
<servlet>
    <servlet-name>servletJSP</servlet-name>
    <jsp-page>/pagina.jsp</jsp-page>
    <init-param>
      <param-name>nome</param-name>
      <param-value>guest</param-value>
    </init-param>
</servlet>
```

(c) request

- Referência para os dados de entrada enviados na requisição do cliente (no GET ou POST, por exemplo, em HTTP)
  - É um objeto do tipo javax.servlet.http.HttpServletRequest
- Usado para
  - Guardar e recuperar atributos que serão usadas enquanto durar a requisição (que pode durar mais de uma página)
  - Recuperar parâmetros passados pelo cliente (dados de um formulário HTML, por exemplo)
  - Recuperar cookies
  - Descobrir o método usado (GET, POST)

```
String method = request.getMethod();
```

(c) exemplos

- *URL no browser:*

```
http://servidor/programa.jsp?nome=Fulano&id=5
```

- *Recuperação dos parâmetros no programa JSP:*

```
<%
String nome = request.getParameter("nome");
String idStr = request.getParameter("id");
int id = Integer.parseInt(idStr);
%>
<p>Bom dia <%=nome %>! (cod: <%=id %>
```

- *Cookies*

```
Cookie[] c = request.getCookies()
```

(d) response

- Referência aos dados de saída enviados na resposta do servidor enviada ao cliente
  - É um objeto do tipo javax.servlet.http.HttpServletResponse
- Usado para
  - Definir o tipo dos dados retornados (default: text/thml)
  - Criar cookies

```
Cookie c = new Cookie("nome", "valor");
response.addCookie(c);
```

  - Definir cabeçalhos de resposta
  - Redirecionar

```
response.sendRedirect("pagina2.html");
```

(e) out

- *Representa o stream de saída da página (texto que compõe o HTML que chegará ao cliente).*
  - *É instância da classe javax.servlet.jsp.JspWriter (implementação de java.io.Writer)*
- *Equivalente a* `response.getWriter();`
- *Principais métodos*

  `print()` e `println()` - *imprimem Unicode*
- *Os trechos de código abaixo são equivalentes*

```
<% for (int i = 0; i < 10; i++) {
out.print("<p> Linha " + i);
} %>

<% for (int i = 0; i < 10; i++) { %>
<p> Linha <%= i %>
<% } %>
```

(f) session

- Representa a sessão do usuário
  - O objeto é uma instância da classe *javax.servlet.http.HttpSession*
- Útil para armazenar valores que deverão permanecer durante a sessão (set/getAttribute())

```
Date d = new Date();
session.setAttribute("hoje", d);
  ...
Date d = (Date)
          session.getAttribute("hoje");
```

(g) application

- Representa o contexto ao qual a página pertence
  - Instância de javax.servlet.ServletContext
- Útil para guardar valores que devem persistir pelo tempo que durar a aplicação (até que o servlet seja descarregado do servidor)
- Exemplo

```
Date d = new Date();
application.setAttribute("hoje", d);
  ...
Date d = (Date)
  application.getAttribute("hoje");
```

(h) pageContext

- Instância de javax.servlet.jsp.PageContext
- Oferece acesso a todos os outros objetos implícitos.
  Métodos:
  - getPage() - retorna page
  - getRequest() - retorna request
  - getResponse() - retorna response
  - getOut() - retorna out
  - getSession() - retorna session
  - getServletConfig() - retorna config
  - getServletContext() - retorna application
  - getException() - retorna exception
- Constrói a página (mesma resposta) com informações localizadas em outras URLs
  - pageContext.forward(String) - mesmo que ação <jsp:forward>
  - pageContext.include(String) - mesmo que ação <jsp:include>

Escopo dos objetos

- A persistência das informações depende do escopo dos objetos onde elas estão disponíveis
- Constantes da classe javax.servlet.jsp.PageContext identificam escopo de objetos
  - pageContext — PageContext.PAGE_SCOPE
  - request — PageContext.REQUEST_SCOPE
  - session — PageContext.SESSION_SCOPE
  - application — PageContext.APPLICATION_SCOPE

- Métodos de pageContext permitem setar ou buscar atributos em qualquer objeto de escopo:
  - `setAttribute(nome, valor, escopo)`
  - `getAttribute(nome, escopo)`

(i) exception

- *Não existe em todas as páginas - apenas em páginas designadas como páginas de erro*

```
<%@ page isErrorPage="true" %>
```

- *Instância de java.lang.Throwable*
- *Exemplo:*

```
<h1>Ocoreu um erro!</h1>
<p>A exceção é
<%= exception %>
Detalhes: <hr>
<% exception.printStackTrace(out); %>
```

Usando beans

- JavaBeans são objetos escritos de acordo com um determinado padrão que permite tratá-los como componentes de um framework
  - Ótimos para separar os detalhes de implementação de uma aplicação de seus "serviços"
  - Permitem encapsular dados recebidos de outras partes da aplicação e torná-los disponíveis para alteração e leitura através de uma interface uniforme.
- Podem ser usados com JSP para remover grande parte do código Java de uma página JSP
  - Maior facilidade de manutenção e depuração
  - Separação de responsabilidade e reuso de componentes

Como incluir um bean

- Para que um bean possa ser usado por uma aplicação JSP, ele deve estar compilado e localizado dentro do CLASSPATH reconhecido pelo servidor
  - No subdiretório WEB-INF/classes do seu contexto
- Para incluir:

```
<jsp:useBean id="nome_da_referência"
             class="pacote.NomeDaClasse"
             scope="page|session|request|application">
</jsp:useBean>
```

- O atributo de escopo é opcional e indica o tempo de vida do Java Bean. Se omitido, será page, que o limita à página
  - Com escopo de request, o bean pode ser recuperado com outra instrução <jsp:useBean> que esteja em outra página que receber a mesma requisição (via dispatcher)
  - Com escopo de session, o bean é recuperável em páginas usadas pelo mesmo cliente, desde que <%@page> não tenha session=false

Como incluir um bean

- *O nome do bean (atributo id) comporta-se como uma referência a um objeto Java*
- *Incluir o tag*

```
<jsp:useBean id="bean" class="bean.HelloBean"
             scope="request" />
```

é o mesmo que incluir na página

```
<% Object obj = request.getAttribute("bean");
   bean.HelloBean bean = null;
   if (obj == null) {
      bean = new bean.HelloBean();
      request.setAttribute("bean", bean);
   } else {
      bean = (bean.HelloBean) obj;
   } %>
```

- *O id pode ser usado em scriptlets para usar membros do bean*

```
<%  bean.setValor(12); %>
```

Propriedades

- *JavaBeans possuem propriedades que podem ser somente-leitura ou leitura-alteração.*
- *O nome da propriedade é sempre derivada do nome do método getXXX():*

```
public class Bean {
  private String mensagem;
  public void setTexto(String x) {
    mensagem = x;
  }
  public String getTexto() {
    return mensagem;
  }
}
```

- *O bean acima tem uma propriedade (RW) chamada* **texto**

Propriedades

- *Páginas JSP podem ler ou alterar propriedades de um bean usando os tags*

```
<jsp:setProperty name="bean" property="propriedade"
                 value="valor"/>
```

que equivale a `<% bean.setPropriedade(valor); %>` e

```
<jsp:getProperty name="bean" property="propriedade"/>
```

que equivale a `<%=bean.getPropriedade() %>`

- *Observe que o nome do bean é passado através do atributo name, que corresponde ao atributo id em* `<jsp:useBean>`
- *Valores são convertidos de e para String automaticamente*
- *Parâmetros HTTP com mesmo nome que as propriedades têm valores passados automaticamente com <jsp:setProperty>*
  - *Se não tiverem, pode-se usar atributo param de <jsp:setProperty>*
  - *<jsp:setProperty ... property="*"/> lê todos os parâmetros*

Inicialização de beans

- *A tag <jsp:useBean> simplesmente cria um bean chamando seu construtor. Para inicializá-lo, é preciso chamar seus métodos setXXX() ou usar <jsp:setProperty> após a definição*

```
<jsp:useBean id="bean" class="bean.HelloBean" />
<jsp:setProperty name="bean" property="prop" value="valor"/>
```

- *Se um bean já existe, porém, geralmente não se deseja inicializá-lo.*
- *Neste caso, a inicialização pode ser feita dentro do marcador <jsp:useBean> e o sistema só a executará se o bean for novo (se já existir, o código será ignorado)*

```
<jsp:useBean id="bean" class="bean.HelloBean" />
  <jsp:setProperty name="bean" property="prop" value="valor"/>
</jsp:useBean>
```

ou

```
<jsp:useBean id="bean" class="bean.HelloBean" />
  <% bean.setProp(valor); %>
</jsp:useBean>
```

Matando beans

- *Beans são sempre gravados em algum objeto de escopo: page, request, session ou application*
  - *Persistem até que o escopo termine ou expirem devido a um timeout (no caso de sessões)*
- *Para se livrar de beans persistentes, use os métodos removeAttribute(), disponíveis para cada objeto de escopo:*

```
session.removeAttribute(bean);
application.removeAttribute(bean);
request.removeAttribute(bean);
```

Composite View

- *Páginas complexas geralmente possuem diversas seções independentes*
  - *Menus*
  - *Corpo da página*
  - *Trechos altamente dinâmicos (notícias, etc.)*
- *Em vez de usar um arquivo único, uma página complexa pode ser formada a partir da composição de textos menores (Composite View patttern)*
- *JSP oferece duas soluções*
  - *Inclusão estática (no momento da compilação do servlet)*
  - *Inclusão dinâmica (no momento da requisição)*

Inclusão estática

- *Mais eficiente: fragmentos são incluídos em único servlet*
- *Indicada quando estrutura não muda com freqüência (conteúdo pode mudar)*
  - *Menus, Logotipos e Avisos de copyright*
  - *Telas com miniformulários de busca*
- *Implementada com <%@ include file="fragmento" %>*

```
<!-- Menu superior -->
<table>
<tr><td><%@ include file="menu.jsp" %></td></tr>
</table>
<!-- Fim do menu superior -->
```

Fragmento menu.jsp

```
<a href="link1">Item 1</a></td>
<td><a href="link2">Item 2</a></td>
<a href="link3">Item 3</a>
```

Se tela incluida contiver novos fragmentos, eles serão processados recursivamente

Inclusão dinâmica

- *Mais lento: fragmentos não são incluídos no servlet mas carregados no momento da requisição*
- *Indicada para blocos cuja estrutura muda com freqüência*
  - *Bloco central ou notícias de um portal*
- *Implementada com <jsp:include page="fragmento"/>*
- *Pode-se passar parâmetros em tempo de execução usando <jsp:param> no seu interior*

```
<!-- Texto principal -->
<table>
<tr><td>
<jsp:include page="texto.jsp">
   <jsp:param name="data" value="<%=new Date() %>">
</jsp:include>
</td></tr> </table>
<!-- Fim do texto principal -->
```

Repasse de requisições

- *Uma requisição pode ser repassada de uma página JSP para outra página ou servlet usando RequestDispatcher*

```
<% RequestDispatcher rd =
            request.getRequestDispatcher("url");
   rd.dispatch(request, response);
%>
```

- *O mesmo efeito é possível sem usar scriptlets com a ação padrão <jsp:forward>*
- *Assim como <jsp:include>, pode incluir parâmetros recuperáveis na página que receber a requisição usando request.getParameter() ou <jsp:getProperty> se houver bean*

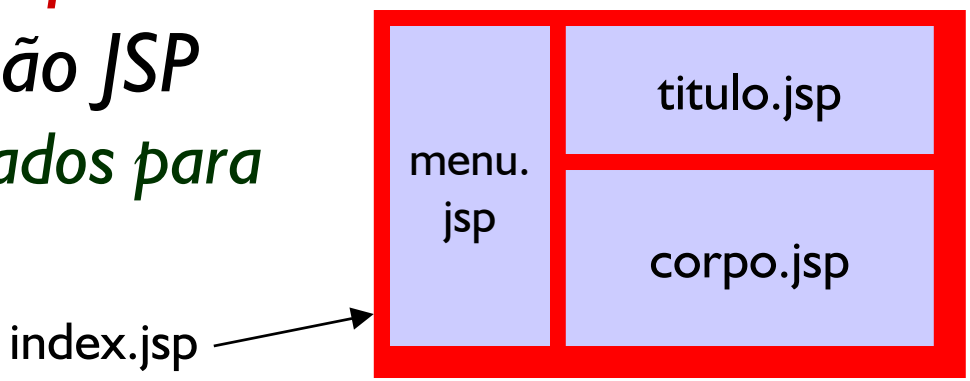
```
<% if (nome != null) { %>
<jsp:forward page="segunda.jsp">
   <jsp:param name="nome" value="<%=nome %>">
</jsp:forward>
<% } %>
```

Exercícios

- 1. Escreva um JSP data.jsp que imprima a data de hoje.
  - Use Calendar e GregorianCalendar
- 2. Escreva um JSP temperatura.jsp que imprima uma tabela HTML de conversão Celsius-Fahrenheit entre -40 e 100 graus Celsius com incrementos de 10 em 10
  - A fórmula é $F = 9/5\ C + 32$
- 3. Altere o exercício anterior para que a página também apresente um campo de textos para entrada de temperatura em um formulário que envie os dados com POST. Faça com que a própria página JSP receba a mensagem
  - a) Identifique, no início, o método com request.getMethod() (retorna POST ou GET, em maiúsculas).
  - b) Se o método for POST, mostre, em vermelho, antes da exibição do formulário, o texto: "x graus F = y graus C" onde x é o valor digitado pelo usuário e y é a resposta.

Exercícios

- *4. Use um JavaBean para implementar a aplicação temperatura.jsp do exercício anterior*
  - *O bean deve ter as propriedades celsius e fahrenheit*
  - *Ao definir uma das propriedades, a outra deve automaticamente conter o valor convertido*
  - *Use o bean na página que imprime a tabela*
- *5. Monte a página ao lado usando os arquivos em cap7/composite e os marcadores de inclusão JSP*
  - *Escolha os mais apropriados para cada situação*



helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)

© 1999-2003, Helder da Rocha

Java 2 Enterprise Edition

Fundamentos de Taglibs, MVC e Struts

Helder da Rocha
www.argonavis.com.br

Objetivos

- Este módulo tem como objetivo apenas apresentar os fundamentos básicos de taglibs e abordar superficialmente frameworks MVC
  - Uma abordagem mais profunda foge do escopo deste curso, que não trata apenas de aplicações Web.
  - Para maior aprofundamento, consulte os exemplos de aplicações incluídas no CD (cap08) e as referências no final desta apresentação.
- Este módulo é opcional
  - Para mais detalhes, consulte os slides dos cursos J550 (JSP e servlets), J551 (Struts) e J931 (Design Patterns J2EE)

Sumário

- 1. Custom tags
  - O que são taglibs e custom tags?
  - Como usar e criar custom tags
  - Tipos de tags e principais componentes
  - Fundamentos de JSTL
- 2. Design de aplicações Web
  - Aplicações JSP procedurais: Model 1
  - Arquitetura MVC e JSP Model 2
  - Usando Struts para implementar JSP Model 2

1. Custom tags

- *JSP com JavaBeans fornecem um meio de diminuir código Java da página, mas não totalmente*
  - *Designers de página ainda têm que usar elementos de script para loops e lógica condicional (getProperty e setProperty não bastam)*
  - *Nem sempre os JavaBeans são suficientes para encapsular toda a lógica da aplicação*
- *A especificação prevê a criação de elementos XML personalizados (custom tags) para resolver essas limitações*
  - *Organizados em bibliotecas (taglibs)*
  - *Cada biblioteca tem seu próprio namespace*
- *Taglibs são declaradas no início de cada página ...*

```
<%@taglib uri="http://abc.com/ex" prefix="exemplo"%>
```

namespace

- *... e usadas em qualquer lugar*

```
<exemplo:dataHoje />
```

produz → **Tuesday, May 5, 2002 13:13:13 GMT-03**

Como usar custom tags

- A URI usada para identificar o prefixo de um custom tag não precisa ser real (e apontar para um local)
  - Serve apenas como **identificador**
  - Ligação entre a especificação da biblioteca (arquivo TLD) e o identificador é feito no arquivo web.xml

```
<web-app>
    ...
    <taglib>
        <taglib-uri>http://abc.com/ex</taglib-uri>
        <taglib-location>
        /WEB-INF/mytaglib.tld
        </taglib-location>
    </taglib>
</web-app>
```

Este é o deployment descriptor do Taglib.

Localização real!

Exemplo de arquivo TLD

```
<?xml version="1.0" ?>
<!DOCTYPE taglib PUBLIC
"-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>exemplo</short-name>
    <uri>http://abc.com/ex</uri>

    <tag>
        <name>dataHoje</name>
        <tag-class>exemplos.DateTag</tag-class>
        <description>Data de hoje</description>
    </tag>

</taglib>
```

Sugestão de prefixo (autor de página pode escolher outro na hora)

URI identifica o prefixo. (autor de página tem que usar exatamente esta URI)

Implementação

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class DateTag extends TagSupport {
    /**
     * Chamado quando o tag terminar.
     */
    public int doEndTag() throws JspException {
        try {
            Writer out = pageContext.getOut();
            java.util.Date = new java.util.Date();
            out.println(hoje.toString());
        } catch (java.io.IOException e) {
            throw new JspException (e);
        }
        return Tag.EVAL_PAGE;
    }
}
```

Para tags que não precisam processar o corpo use a interface Tag ou sua implementação TagSupport

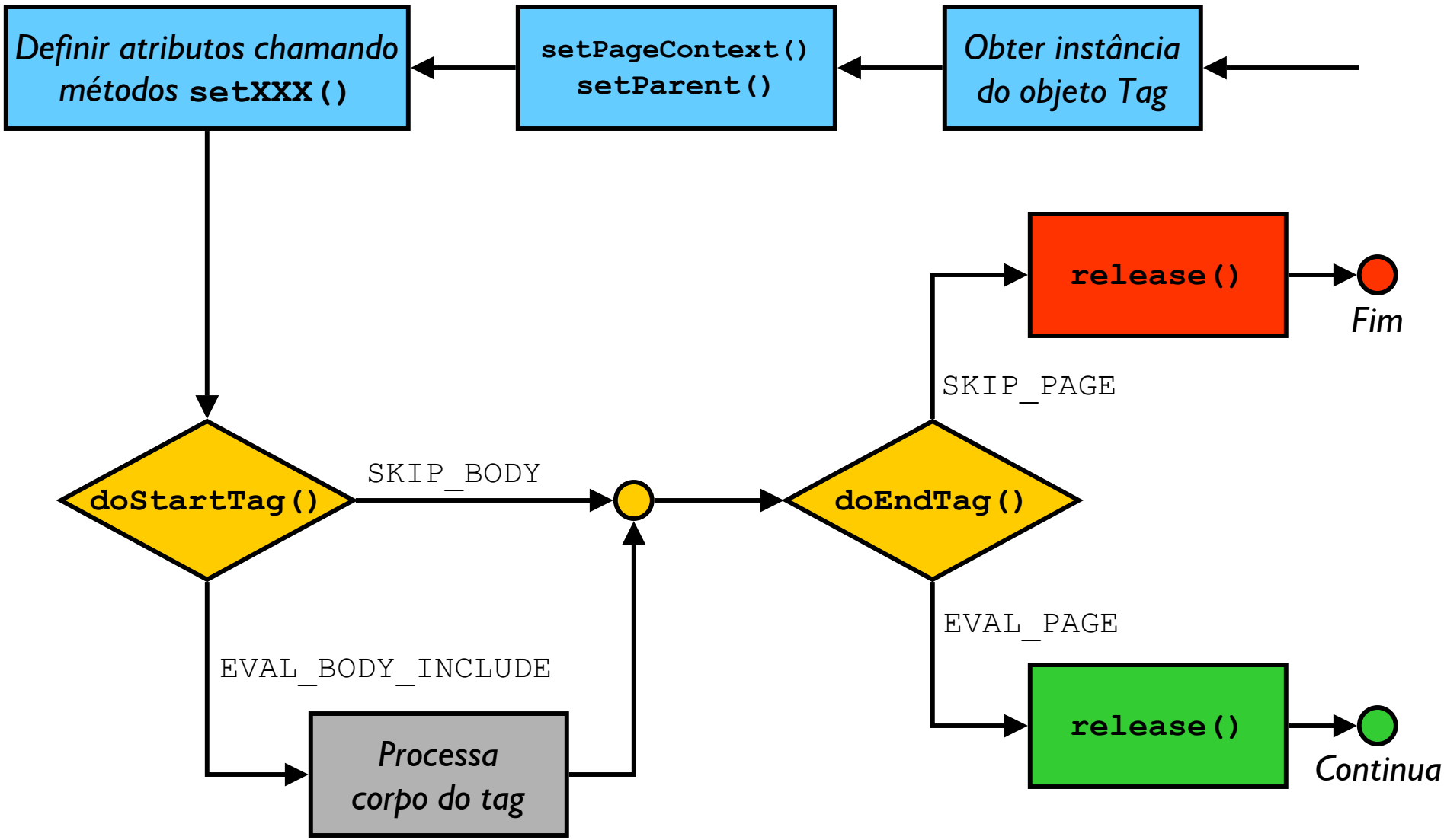
Use doStartTag() para processamento antes do tag, se necessário

Para este método, pode ser EVAL_PAGE ou SKIP_PAGE

Tipos de tags

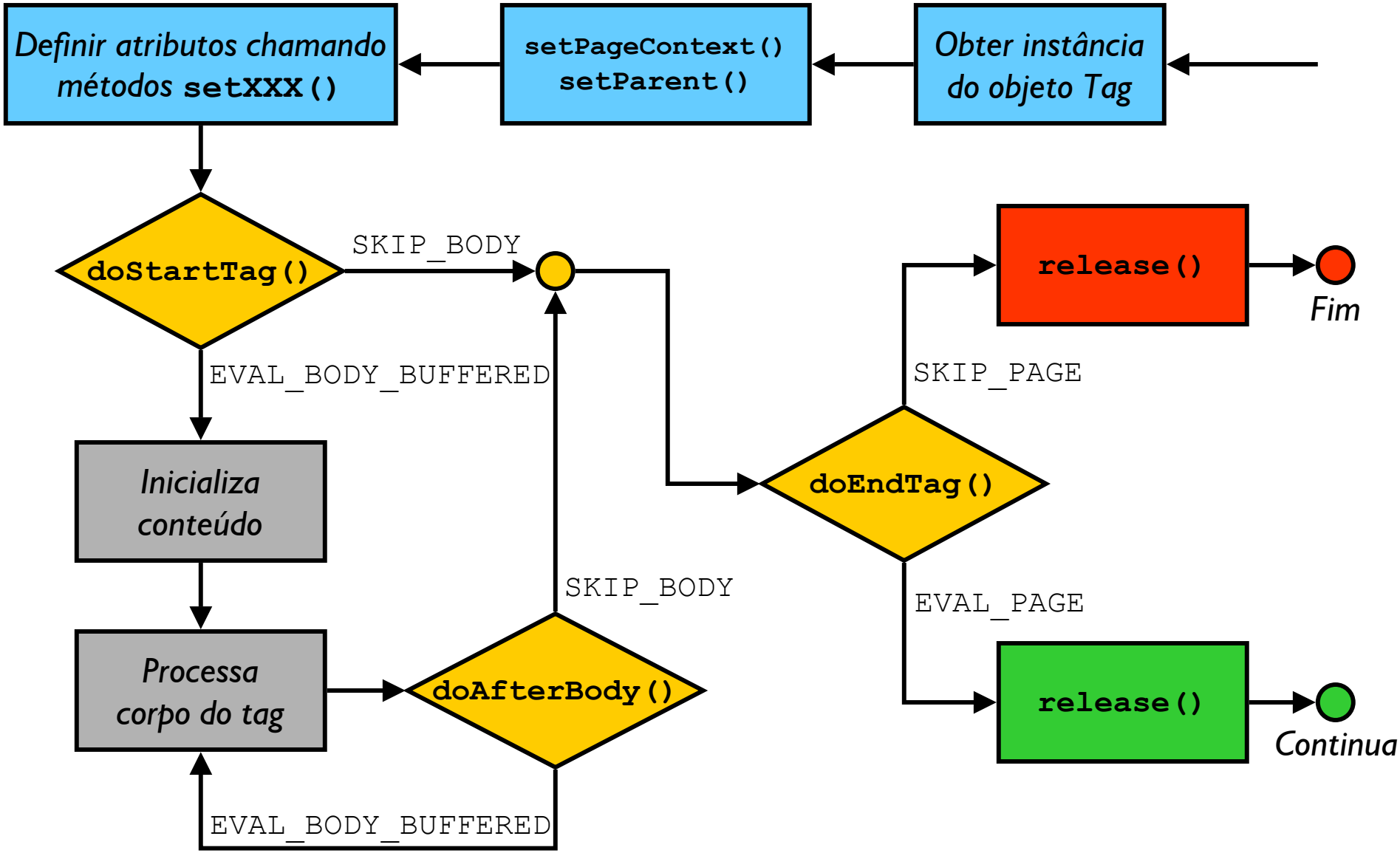
- *Há vários tipos de custom tags. Cada estratégia utiliza diferentes classes base e métodos*
- *1. Diferenciados por herança:*
  - *Tags simples: implementam a interface Tag (TagSupport é uma implementação neutra).*
  - *Tags com corpo que requer processamento: implementam BodyTag (BodyTagSupport é implementação neutra)*
- *Diferenciados por outras características*
  - *Tags que possuem atributos*
  - *Tags que definem variáveis de scripting fora do seu escopo (requerem classe extra com "Tag Extra Info")*
  - *Tags que interagem com outros tags*

Ciclo de vida de objetos Tag



Fonte: [6]

Ciclo de vida de objetos BodyTag



Fonte: [6]

Atributos

- *Para definir atributos em um tag é preciso*
  1. *Definir método setXXX() com o nome do atributo*
  2. *Declarar atributo no descritor (TLD)*

```
<xyz:upperCase text="abcd" />
```

```
public class UpperCaseTag {
    public String text;
    public void setText(String text) {
        this.text = text;
    } (...)
```

```
<tag> <name>upperCase</name> (...)
    <attribute>
        <name>text</name>
        <required>true</required>
        <rtexprvalue>false</rtexprvalue>
    </attribute> (...) </tag>
```

- *Os atributos devem setar campos de dados no tag*
  - *Valores são manipulados dentro dos métodos doXXX():*

```
public int doStartTag() throws JspException {
    Writer out = pageContext.getOut();
    out.println(text.toUpperCase());   (...)
```

Obtenção do conteúdo do Body

- *O objeto out, do JSP, referencia a instância BodyContent de um tag enquanto processa o corpo*
  - *BodyContent é subclasse de JspWriter*
  - *Tag decide se objeto BodyContent deve ser jogado fora ou impresso (na forma atual ou modificada)*
- *Exemplo*

```
public int doAfterBody() throws JspException {
    BodyContent body = getBodyContent();
    String conteudo = body.getString();
    body.clearBody();
    (...)
    getPreviousOut().print(novoTexto);
}
```

Guarda conteúdo do tag

Apaga conteúdo

Imprime texto na página (e não no body do Tag)

Exemplos de Custom Tags

- *Veja* **exemplos/cap08/taglibs/**
  - *Vários diferentes exemplos de custom tags (do livro [6])*
  - *Código fonte em taglib/src/taglibdemo/*.java*
  - *Páginas exemplo em src/*Test.jsp (6 exemplos)*

1. Configure build.properties, depois, monte o WAR com:

```
> ant build
```

2. Copie o WAR para o diretório webapps do Tomcat

```
> ant deploy
```

3. Execute os tags, acessando as páginas via browser:

```
http://localhost:porta/mut/
```

- *Veja também* **exemplos/cap08/mvc/hellojsp_2/**
  - *Aplicação MVC que usa custom tags (veja como executar na próxima seção)*

JSP Standard Tag Library

- Esforço de padronização do JCP: JSR-152
  - Baseado no Jakarta Taglibs (porém bem menor)
- Oferece dois recursos
  - Conjunto padrão de tags básicos (Core, XML, banco de dados e internacionalização)
  - Linguagem de expressões do JSP 1.3
- Oferece mais controle ao autor de páginas sem necessariamente aumentar a complexidade
  - Controle sobre dados sem precisar escrever scripts
  - Estimula a separação da apresentação e lógica
  - Estimula o investimento em soluções MVC

Como usar JSTL

- *1. Fazer o download da última versão do site da Sun*
- *2. Copiar os JARs das bibliotecas desejadas para o diretório WEB-INF/lib/ da sua aplicação Web e os arquivos TLD para o diretório WEB-INF/*
- *3. Declarar cada taglib e associá-la com seu TLD no deployment descriptor web.xml.*
- *4. Incluir em cada página que usa os tags:*

```
<%@ taglib uri="uri_da_taglib"
           prefix="prefixo" %>
```

- *5. Usar os tags da biblioteca com o prefixo definido no passo anterior*

```
<prefixo:nomeTag atributo="..."> ...
</prefixo:nomeTag>
```

Quatro bibliotecas de tags

- *Core library: tags para condicionais, iterações, urls, ...*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/core"
           prefix="c" />
```

  - *Exemplo:* `<c:if test="..." ... >...</c:if>`
- *XML library: tags para processamento XML*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/xml"
           prefix="x" />
```

  - *Exemplo:* `<x:parse>...</x:parse>`
- *Internationalization library*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/fmt"
           prefix="fmt" />
```

  - *Exemplo:* `<fmt:message key="..." />`
- *SQL library*

```
<%@ taglib uri="http://java.sun.com/jstl/ea/sql"
           prefix="sql" />
```

  - *Exemplo:* `<sql:update>...</sql:update>`

Linguagem de expressões

- *Permite embutir em atributos expressões dentro de delimitadores ${...}*
  - *Em vez de* `request.getAttribute("nome")`
    `${nome}`
  - *Em vez de* `bean.getPessoa().getNome()`
    `${bean.pessoa.nome}`
- *Suporta operadores aritméticos, relacionais e binários*
- *Converte tipos automaticamente*

```
<tag item="${request.valorNumerico}" />
```

- *Valores default*

```
<tag value="${abc.def}" default="todos" />
```

Principais ações

- *Suporte à impressão da linguagem expressões*
  - `<c:out value="${pessoa.nome}" />`
- *Expressões condicionais*
  - `<c:if test="${pessoa.idade >= 18}">`
    `<a href="adultos.html">Entrar</a>`
    `</c:if>`
  - `<c:choose>`
    `<c:when test="${dia.hora == 13}">`
    `<c:out value="${mensagemEspecial}" />`
    `</c:when>`
    `<c:otherwise>`
    `<c:out value="${mensagemPadrao}" />`
    `</c:otherwise>`
  - `</c:choose>`
- *Iteração*
  - `<c:forEach items="${pessoas}" var="p" varStatus="s">`
    `<c:out value="${s.count}"/>. <c:out value="${p}"/>`
    `</c:forEach>`

Internacionalização, XML e SQL

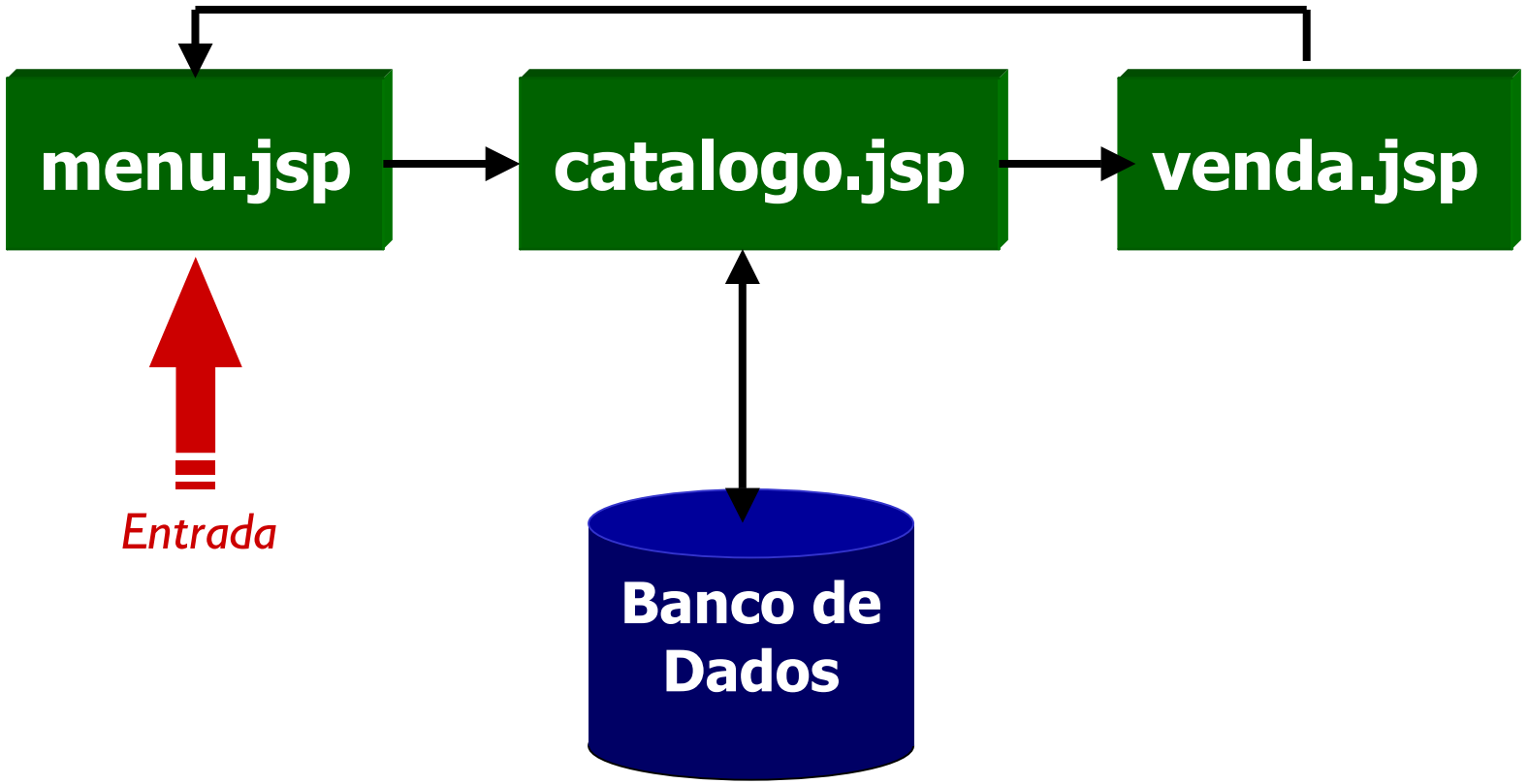
- Ler propriedade de ResourceBundle
  - `<fmt:message key="chave.do.bundle" />`
- Operações diretas em banco de dados
  - `<sql:query dataSource="${dsn}">`
    `SELECT...</sql:query>`
  - `<sql:transaction>`, `<sql:update>`, etc.
- Operações com XML
  - Uso de expressões XPath em tags JSTL para XML
  - Ações XML: `<x:out>`, `<x:set>`, `<x:if>`, `<x:choose>`, `<x:forEach>` (atributo select contém expr. XPath)
  - `<x:parse>` Processa XML usando DOM ou filtro SAX
  - `<x:transform>` Realiza transformação XSLT.

2. Design de aplicações JSP

- *Design centrado em páginas*
  - *Aplicação JSP consiste de seqüência de páginas (com ou sem beans de dados) que contém código ou links para chamar outras páginas*
- *Design centrado em servlet (FrontController* ou MVC)*
  - *Aplicação JSP consiste de páginas, beans e servlets que controlam todo o fluxo de informações e navegação*
  - *Este modelo favorece uma melhor organização em camadas da aplicação, facilitando a manutenção e promovendo o reuso de componentes.*
  - *Um único servlet pode servir de fachada*
  - *Permite ampla utilização de J2EE design patterns*

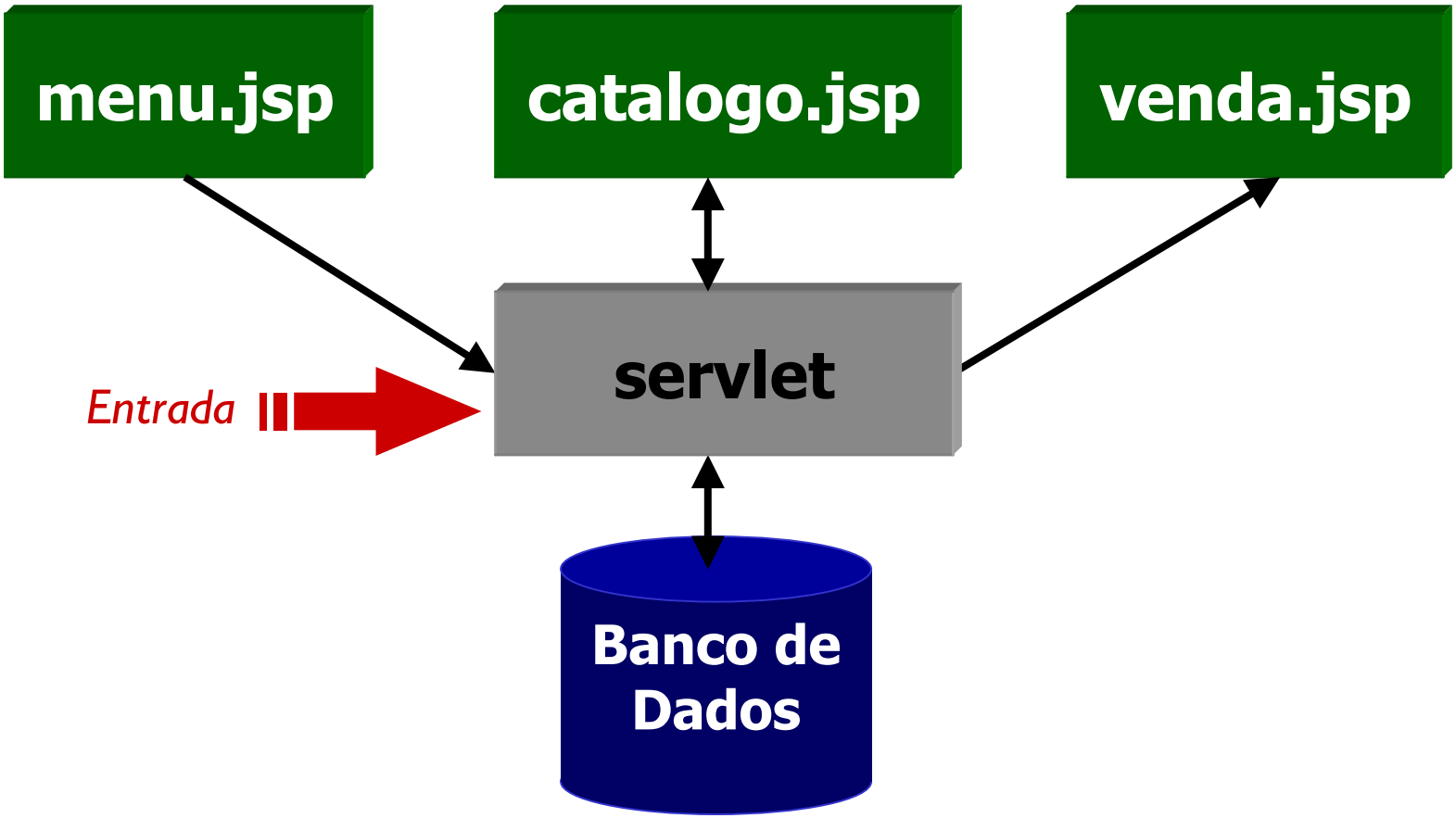
** FrontController é um J2EE design pattern. Vários outros design patterns serão identificados durante esta seção. Para mais informações, veja Sun Blueprints [7]*

Layout centrado em páginas (JSP Model 1)



Fonte: [6]

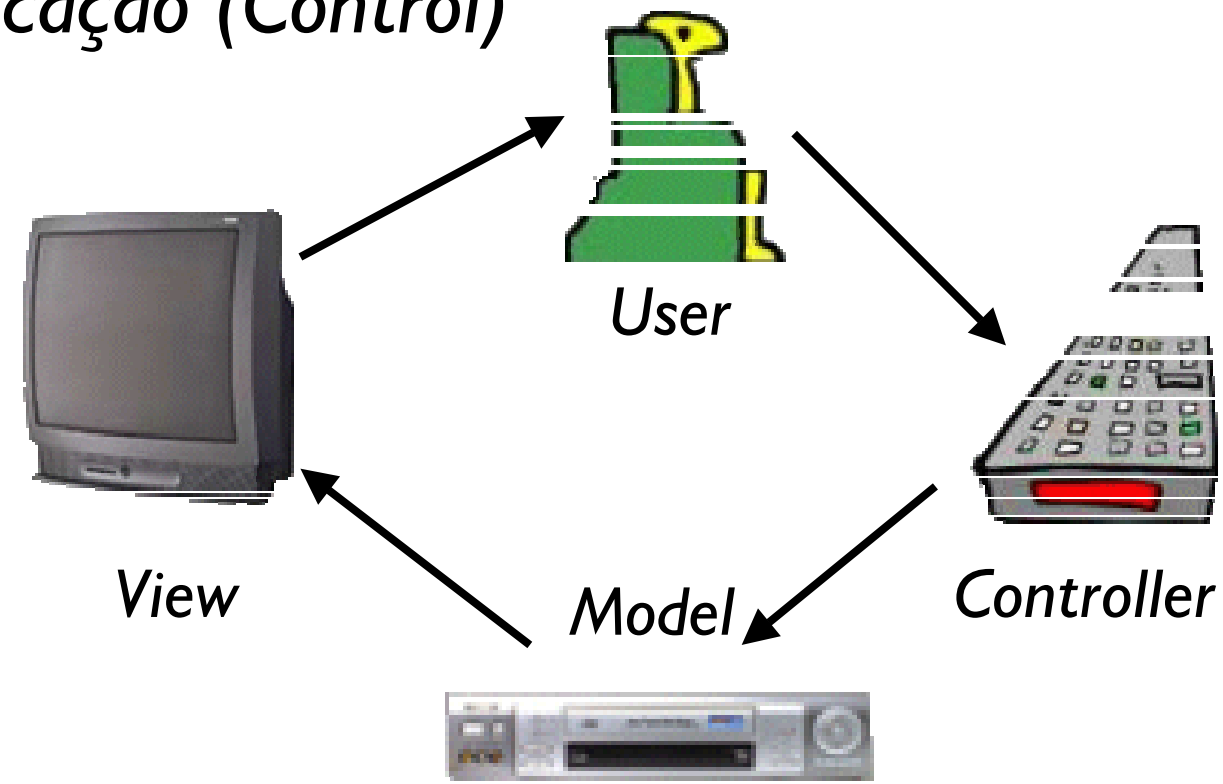
Layout centrado em servlet (JSP Model 2)



Fonte: [6]

O que é MVC

- *Padrão de arquitetura: Model View Controller*
- *Técnica para separar dados ou lógica de negócios (Model) da interface do usuário (View) e do fluxo da aplicação (Control)*



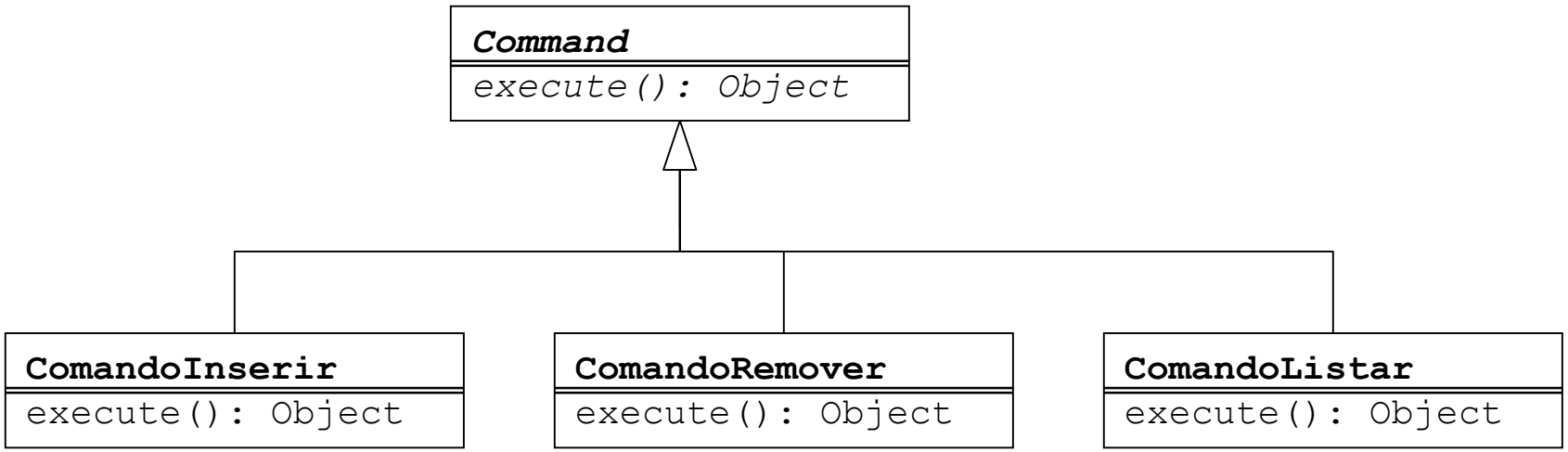
Fonte: *http://www.computer-programmer.org/articles/struts/*

Como implementar?

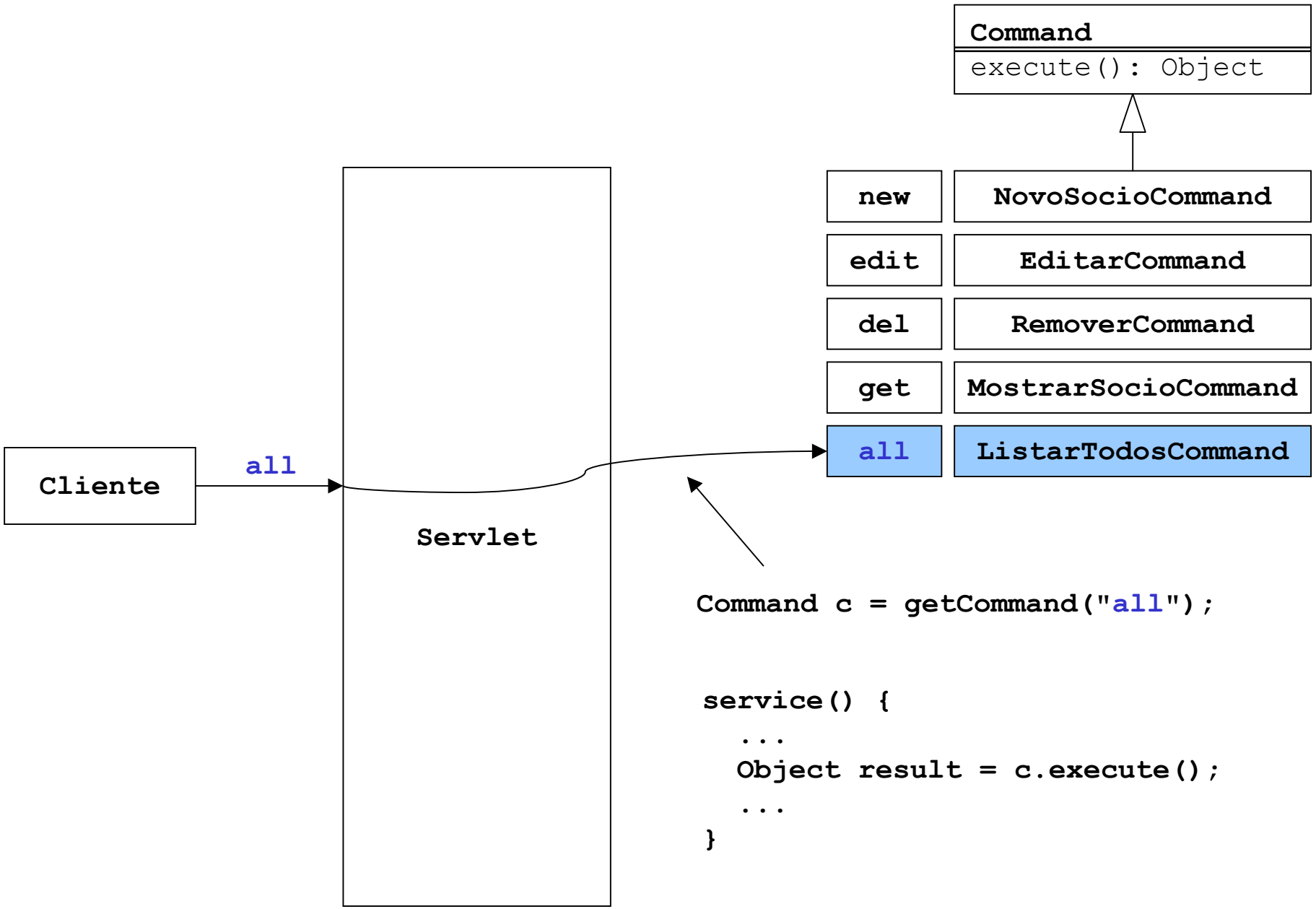
- *Há várias estratégias*
- *Todas procuram isolar*
  - *As operações de controle de requisições em servlets e classes ajudantes,*
  - *Operações de geração de páginas em JSP e JavaBeans, e*
  - *Lógica das aplicações em classes que não usam os pacotes javax.servlet*
- *Uma estratégia consiste em se ter um único controlador (FrontController pattern) que delega requisições a diferentes objetos que implementam comandos que o sistema executa (Command pattern)*

Command Pattern

- *É um padrão de projeto clássico catalogado no livro "Design Patterns" de Gamma et al (GoF = Gang of Four)*
  - *Para que serve: "Encapsular uma requisição como um objeto, permitindo que clientes parametrizem diferentes requisições, filas ou requisições de log, e suportar operações reversíveis." [GoF]*
- *Consiste em usar polimorfismo para construir objetos que encapsulam um comando e oferecer um único método execute() com a implementação do comando a ser executado*



Command Pattern



FrontController com Command Pattern

- *Os comandos são instanciados e guardados em uma base de dados na memória (HashMap, por exemplo)*
  - *Pode-se criar uma classe específica para ser fábrica de comandos*
- *O cliente que usa o comando (o servlet), recebe na requisição o nome do comando, consulta-o no HashMap, obtém a instância do objeto e chama seu método execute()*
  - *O cliente desconhece a classe concreta do comando. Sabe apenas a sua interface (que usa para fazer o cast ao obtê-lo do HashMap*
- *No HashMap*

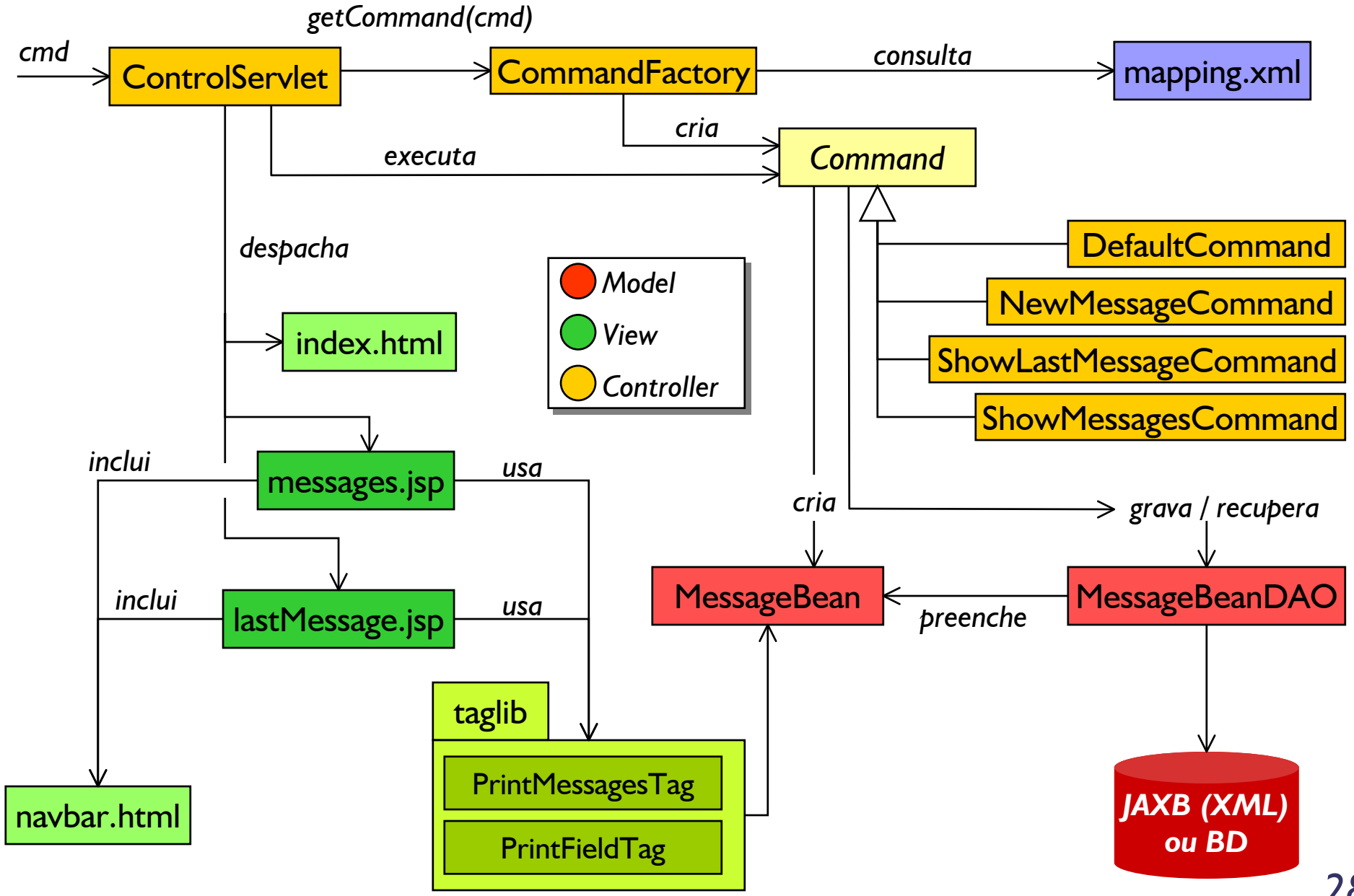
```
Comando c = new ComandoInserir();
comandosMap.put("inserir", c);
```

- *No servlet:*

```
String cmd = request.getParameter("cmd");
Comando c = (Comando)comandosMap.get(cmd);
c.execute();
```

Exemplo de implementação

exemplos/cap08/mvc/hellojsp_2



Mapeamentos de comandos ou ações

- *No exemplo hellojsp_2, o mapeamento está armazendo em um arquivo XML (webinf/mapping.xml)*

```
<command-mapping> (...)
    <command>
        <name>default</name>
        <class>hello.jsp.DefaultCommand</class>
        <success-url>/index.html</success-url>
        <failure-url>/index.html</failure-url>
    </command>
    <command>
        <name>newMessage</name>
        <class>hello.jsp.NewMessageCommand</class>
        <success-url>/lastMessage.jsp</success-url>
        <failure-url>/index.html</failure-url>
    </command>
    <command>
        <name>showAllMessages</name>
        <class>hello.jsp.ShowMessagesCommand</class>
        <success-url>/messages.jsp</success-url>
        <failure-url>/index.html</failure-url>
    </command>
</command-mapping>
```

Comandos ou ações (Service to Worker)

- *Comandos implementam a interface Command e seu método*

```
Object execute(HttpServletRequest request,
               HttpServletResponse response,
               MessageBeanDAO dao);
```

- *Criados por CommandFactory na inicialização e executados por ControlServlet que os obtém via getCommand(nome)*
- *Retornam página de sucesso ou falha (veja mapping.xml)*
- *Exemplo: ShowMessagesCommand:*

```
public class ShowMessagesCommand implements Command {

   public Object execute(...) throws CommandException {
        try {
            MessageBean[] beanArray = dao.retrieveAll();
            request.setAttribute("messages", beanArray);
            return successUrl;
        } catch (PersistenceException e) {
            throw new CommandException(e);
        }
   } (...)
```

Data Access Objects (DAO)

- *Isolam a camada de persistência*
  - *Implementamos persistência JAXB, mas outra pode ser utilizada (SGBDR) sem precisar mexer nos comandos.*
- *Interface da DAO:*

```
public interface MessageBeanDAO {
   public Object getLocator();

   public void persist(MessageBean messageBean)
                              throws PersistenceException;

   public MessageBean retrieve(int key)
                              throws PersistenceException;

   public MessageBean[] retrieveAll()
                              throws PersistenceException;

   public MessageBean retrieveLast()
                              throws PersistenceException;
}
```

Controlador (FrontController)

- *Na nossa aplicação, o controlador é um servlet que recebe os nomes de comandos, executa os objetos que os implementam e repassam o controle para a página JSP ou HTML retornada.*

```
public void service( ..., ... ) ... {
    Command command = null;
    String commandName = request.getParameter("cmd");

    if (commandName == null) {
        command = commands.getCommand("default");
    } else {
        command = commands.getCommand(commandName);
    }

    Object result = command.execute(request, response, dao);
    if (result instanceof String) {
        RequestDispatcher dispatcher =
              request.getRequestDispatcher((String)result);
        dispatcher.forward(request, response);
    }
   ...
}
```

Método de CommandFactory

Execução do comando retorna uma URI

Repassa a requisição para página retornada

ValueBean ViewHelper (Model)

- *Este bean é gerado em tempo de compilação a partir de um DTD (usando ferramentas do JAXB)*

```
public class MessageBean
    extends MarshallableRootElement
    implements RootElement {

    private String _Time;
    private String _Host;
    private String _Message;

    public String getTime() {...}
    public void setTime(String _Time) {...}

    public String getHost() {...}
    public void setHost(String _Host) {...}

    public String getMessage() {...}
    public void setMessage(String _Message) {...}

    ...
}
```

interfaces JAXB permitem que este bean seja gravado em XML (implementa métodos marshal() e unmarshal() do JAXB)

Página JSP (View) com custom tags

- *Página messages.jsp (mostra várias mensagens)*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<%@ taglib uri="/hellotags" prefix="hello" %>
<html>
<head><title>Show All Messages</title></head>
<body>
<jsp:include page="navbar.html" />
<h1>Messages sent so far</h1>
<table border="1">
<tr><th>Time Sent</th><th>Host</th><th>Message</th></tr>

<hello:printMessages array="messages">
    <tr>
        <td><hello:printField property="time" /></td>
        <td><hello:printField property="host" /></td>
        <td><hello:printField property="message" /></td>
    </tr>
</hello:printMessages>

</table>
</body>
</html>
```

Para executar o exemplos

- 1. Mude para exemplos/cap08/mvc/hellojsp_2
- 2. Configure build.properties, depois rode

```
> ant DEPLOY
```

- 3. Inicie o servidor (Tomcat ou JBoss)
- 4. Rode os testes do Cactus

```
> ant RUN-TESTS
```

- 5. Rode a aplicação, acessando a URI

```
http://localhost:porta/hellojsp/
```

- 6. Digite mensagens e veja resultados. Arquivos são gerados em /tmp/mensagens (ou c:\tmp\mensagens)

Struts

O que é Struts?

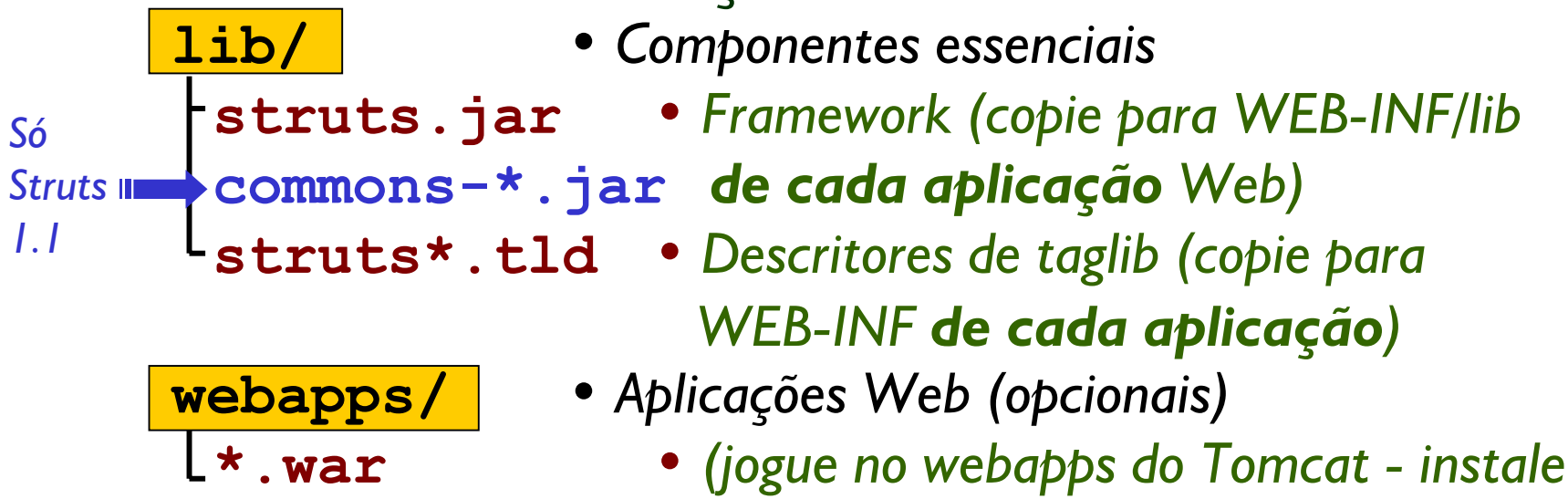
- *Framework para facilitar a implementação da arquitetura MVC em aplicações JSP*
- *Oferece*
  - *Um servlet controlador configurável através de documentos XML externos, que despacham requisições a classes Action (comandos) criadas pelo desenvolvedor*
  - *Uma vasta coleção de bibliotecas de tags JSP (taglibs)*
  - *Classes utilitárias que oferecem suporte a tratamento de XML, preenchimento de JavaBeans e gerenciamento externo do conteúdo de interfaces do usuário*
- *Onde obter: jakarta.apache.org/struts*

Componentes MVC no Struts

- Model (M)
  - Geralmente um objeto Java (JavaBean)
- View (V)
  - Geralmente uma página HTML ou JSP
- Controller (C)
  - org.apache.struts.action.ActionServlet ou subclasse
- Classes ajudantes
  - FormBeans: encapsula dados de forms HTML (M)
  - ActionErrors: encapsulam dados de erros (M)
  - Custom tags: encapsulam lógica para apresentação (V)
  - Actions: implementam lógica dos comandos (C)
  - ActionForward: encapsulam lógica de redirecionamento (C)

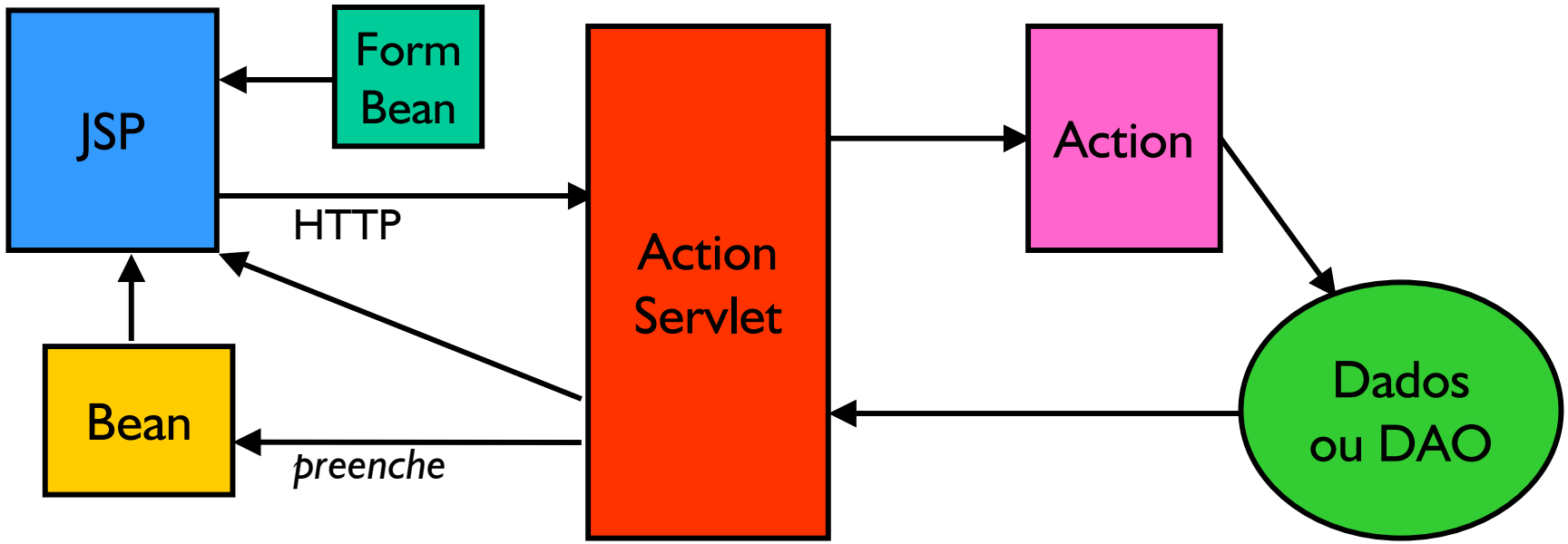
Componentes da distribuição

- Requisitos
  - J2SDK 1.4 ou J2SDK1.3 + JAXP
  - Servlet container, servlet.jar e Jakarta Commons (Struts 1.1)
- Distribuição binária (pré-compilada)
  - Abra o ZIP da distribuição. Conteúdo essencial:



Como funciona?

- *Principais componentes*
  - *ActionServlet: despachante de ações*
  - *Action: classe estendida por cada ação (comando) a ser implementada (usa Command design pattern)*
  - *struts-config.xml: arquivo onde se define mapeamentos entre ações, páginas, beans e dados*



Como instalar

- 1. Copiar os arquivos necessários para sua aplicação
  - Copie lib/struts.jar e lib/commons-*.jar para weu WEB-INF/lib (não coloque no common/lib do Tomcat ou no jre/lib/ext do JDK ou o struts não achará suas classes!)
  - Copie os TLDs das bibliotecas de tags que deseja utilizar para o WEB-INF de sua aplicação (copie todos)
- 2. Para usar o servlet controlador (MVC)
  - Defina-o como um <servlet> no seu web.xml
  - Crie um arquivo WEB-INF/struts.config.xml com mapeamentos de ações e outras as configurações
- 3. Para usar cada conjunto de taglibs
  - Defina, no seu web.xml, cada taglib a ser instalada
  - Carregue a taglib em cada página JSP que usá-la

Configuração do controlador no web.xml

- *Acrescente no seu web.xml*

```
<servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>
      org.apache.struts.action.ActionServlet
   </servlet-class>
   <init-param>
      <param-name>config</param-name>
      <param-value>
         /WEB-INF/struts-config.xml
      </param-value>
   </init-param>
   ... outros init-param ...
</servlet>
```

- *Acrescente também os <servlet-mapping> necessários*
- *Crie e configure as opções de struts-config.xml*
- *Veja nos docs: /userGuide/building_controller.html*
  - *use os arquivos de struts-example.war para começar*

Configuração das Taglibs

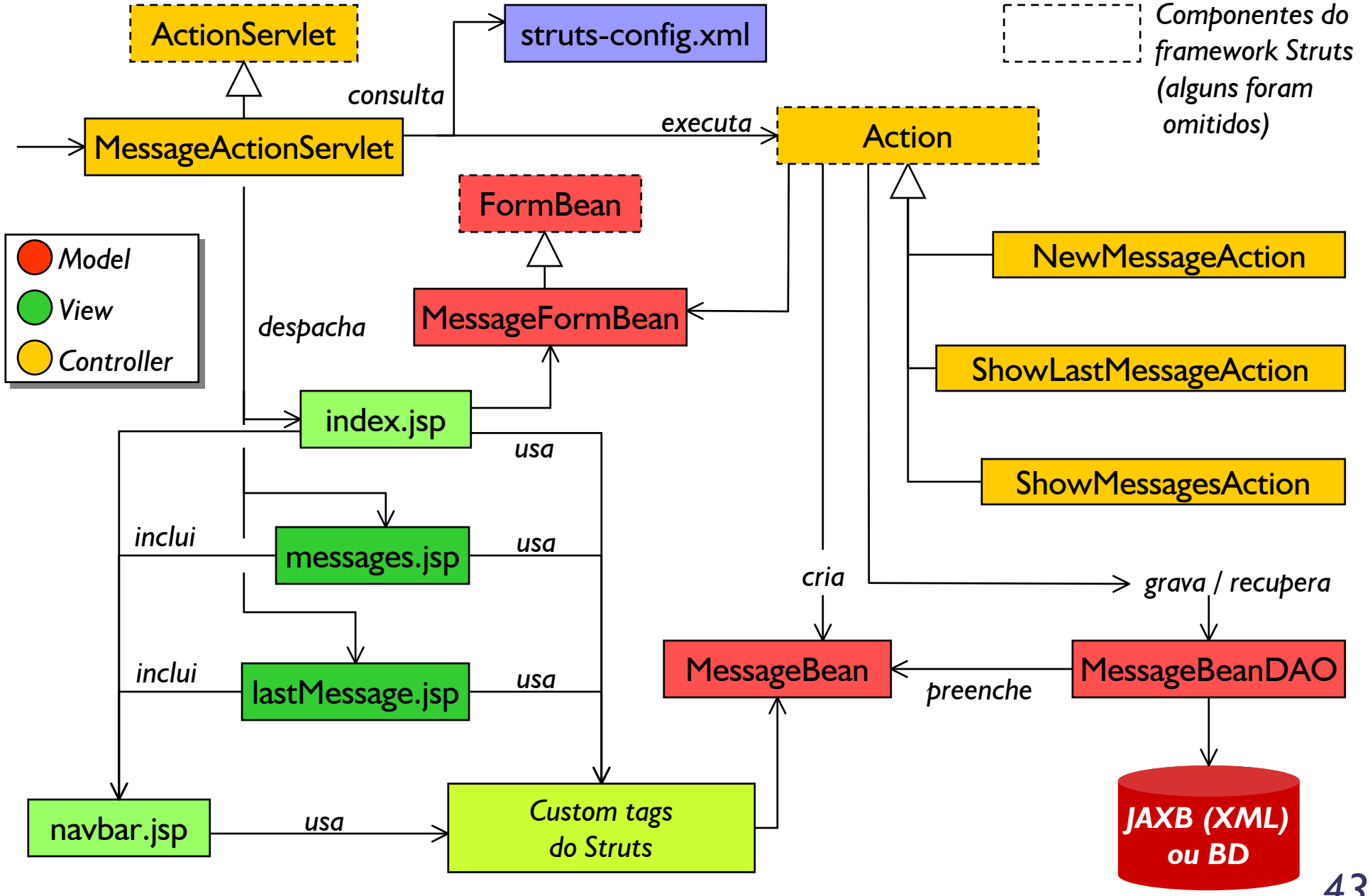
- *Acrescente em **web.xml***
- *Veja detalhes na aplicação struts-example.war ou nos docs:*
  /userGuide/building_controller.html#dd_config_taglib

```
<taglib>
   <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-bean.tld
   </taglib-location>
</taglib>
<taglib>
   <taglib-uri>/WEB-INF/struts-form.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-form.tld
   </taglib-location>
... outros taglibs ...
</taglib>
```

- *Acrescente em cada página JSP*

```
<@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
   ...
```

Implementação de hellojsp com Struts



Mapeamentos (ActionMappings)

- *Veja webinf/struts-config.xml*

```
<struts-config>
  <form-beans>
    <form-bean name="newMessageForm" type="hello.jsp.NewMessageForm" />
  </form-beans>
  <global-forwards>
    <forward name="default" path="/index.jsp" />
  </global-forwards>

  <action-mappings>
    <action path="/newMessage" type="hello.jsp.NewMessageAction"
            validate="true"
            input="/index.jsp" name="newMessageForm" scope="request">
      <forward name="success" path="/showLastMessage.do" />
    </action>
    <action path="/showLastMessage"
            type="hello.jsp.ShowLastMessageAction" scope="request">
      <forward name="success" path="/lastMessage.jsp" />
    </action>
    <action path="/showAllMessages"
            type="hello.jsp.ShowMessagesAction" scope="request">
      <forward name="success" path="/messages.jsp" />
    </action>
  </action-mappings>

  <message-resources parameter="hello.jsp.ApplicationResources" />
</struts-config>
```

FormBeans

- *Form beans permitem simplificar a leitura e validação de dados de formulários*
  - *Devem ser usados em conjunto com custom tags da biblioteca <html:* />*

```
<html:form action="/newMessage" name="newMessageForm"
           type="hello.jsp.NewMessageForm">
    <p>Message: <html:text property="message" />
       <html:submit>Submit</html:submit>
    </p>
</html:form>
```

Configuração em struts-config.xml

```
public class NewMessageForm extends ActionForm {
    private String message = null;
    public String getMessage() { return message; }
    public void setMessage(String message) {
        this.message = message;
    }
    public void reset(...) {
        message = null;
    }
    public ActionErrors validate(...) {...}
}
```

ActionErrors

- *ActionErrors encapsulam erros de operação, validação, exceções, etc.*
  - *Facilitam a formatação e reuso de mensagens de erro.*
- *Exemplo: Método validate() do form bean:*

```
public ActionErrors validate(ActionMapping mapping,
                             HttpServletRequest request) {
  ActionErrors errors = new ActionErrors();
  if ( (message == null) || (message.trim().length() == 0) ) {
    errors.add("message",
               new ActionError("empty.message.error"));
  }
  return errors;
}
```

Nome de campo <input> ao qual o erro se aplica.

Este valor corresponde a uma chave no ResourceBundle

- *Como imprimir:*

```
<html:errors />
```

i18n

- *Informações localizadas podem ser facilmente extraídas de Resource Bundles através de*

```
<bean:message key="chave" />
```

  - *Locale default é usado automaticamente (pode ser reconfigurado)*
- *Exemplo de ResourceBundle*

```
empty.message.error=<tr><td>Mensagem não pode ser
vazia ou conter apenas espaços em branco.</td></tr>
new.message.input.text=Digite a sua mensagem
message.submit.button=Enviar Mensagem
```

hello/jsp/ApplicationResources_pt.properties

- *Configuração em struts-config.xml*

```
<message-resources
      parameter="hello.jsp.ApplicationResources" />
```

- *Exemplo de uso:*

```
<p><bean:message key="new.message.input.text" />
```

Action (Controller / Service To Worker)

- *Controlador processa comandos chamando o método execute de um objeto Action*

```
public class ShowMessagesAction extends Action {

  private String successTarget = "success";
  private String failureTarget = "default";

  public ActionForward execute(ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response)
                          throws IOException, ServletException {
    try {
       MessageBeanDAO dao =
          (MessageBeanDAO)request.getAttribute("dao");
       MessageBean[] beanArray = dao.retrieveAll();
       request.setAttribute("messages", beanArray);
       return (mapping.findForward(successTarget));
     } catch (PersistenceException e) {
       throw new ServletException(e);
     }
  } ...
```

Como rodar o exemplo

- *1. Mude para exemplos/cap08/mvc/hellojsp_3*
- *2. Configure build.properties, depois rode*

```
> ant DEPLOY
```

- *3. Inicie o servidor (Tomcat ou JBoss)*
- *4. Rode os testes do Cactus*

```
> ant RUN-TESTS
```

- *5. Rode a aplicação, acessando a URI*

```
http://localhost:porta/hellojsp-struts/
```

- *6. Digite mensagens e veja resultados. Arquivos são gerados em /tmp/mensagens (ou c:\tmp\mensagens)*

Exercícios

- 1. Escreva um custom tag que
  - Receba como atributos: o nome de uma cor HTML (red, blue, etc.) e um número
  - Receba um texto como corpo
- Sua execução deve imprimir o texto recebido na cor selecionada e no tamanho em pontos recebido.
  - Use CSS:

    `<span style="color: red; font-size: 10pt">texto</span>`

Fontes

[1] *http://www.computer-programmer.org/articles/struts/*

[2] *Manual do Struts* *Copie o arquivo struts-documentation.war (se quiser, altere o nome do WAR antes) para o diretório webapps/ to Tomcat 4.0 ou deploy do JBoss.*

[3] *Jim Farley, Java Enterprise in a Nutshell. O'Reilly, 2002.* *Contém tutorial curso e objetivo sobre JSP e taglibs.*

[4] *Fields/Kolb. Web Development with JavaServer Pages. Manning, 2000.* *Contém dois capítulos dedicados a taglibs básicos e avançados.*

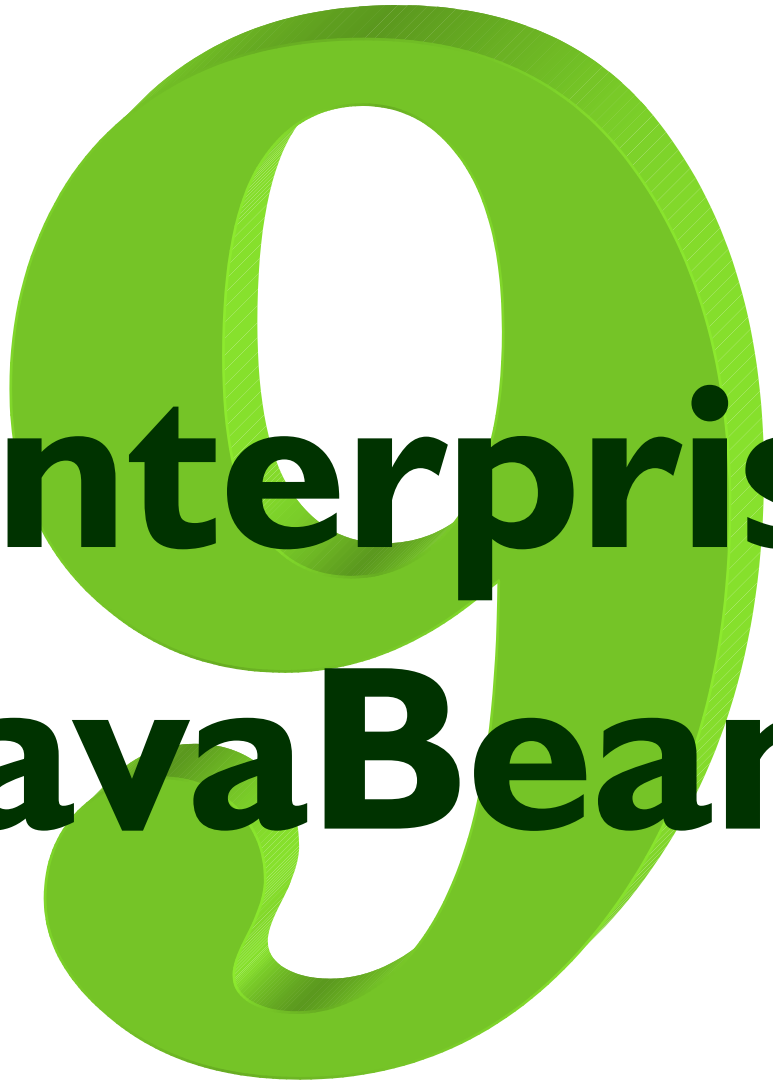
[5] *Sun J2EE Blueprints.*

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)

© 1999-2003, Helder da Rocha



Enterprise JavaBeans

Helder da Rocha
www.argonavis.com.br

Sumário

- Este módulo é uma breve introdução aos conceitos comuns a todos os EJBs

- Por que EJB?
  - Como funciona o middleware implícito
- O que é um Enterprise Bean
  - Componentes de um EJB
  - EJB Object
  - EJB Home
  - Deployment Descriptor
  - Arquivos do fabricante

Por que EJB? (I)

- Sistemas de objetos distribuídos têm em comum uma arquitetura que consiste de
  - O objeto remoto
  - Um stub que representa o objeto remoto no cliente
  - Um esqueleto, que representa o cliente no servidor

- O cliente conversa com o stub pensando tratar-se do próprio objeto
- O esqueleto conversa com o objeto remoto que pensa que é o cliente quem fala com ele
- O stub e o objeto remoto implementam a **mesma interface**!

Fonte da ilustração: Ed Roman, Mastering EJB 2

Por que EJB? (2)

- Desenvolver um sistema distribuído não é simples.
  - Exige se concentrar em aspectos da aplicação que nada tem a ver com o problema de negócio a ser solucionado
  - Exige a observação de condições que geralmente são menos importantes ou dispensáveis em aplicações locais
- É preciso se preocupar com
  - A performance, que pode sofrer abalos em rede
  - Os custos de uma possível ampliação da capacidade
  - A segurança dos dados, controle de acesso, permissões
  - Transações: tarefas poderão falhar no meio do caminho
  - Integridade dos dados: clientes irão acessá-los simultaneamente
  - O que acontecerá se parte do sistema sair do ar?
- Mas esses são problemas de qualquer sistema distribuído
  - Será que precisamos implementar tudo isto?
  - Será que não estaremos reinventando a roda?

Por que EJB? (2)

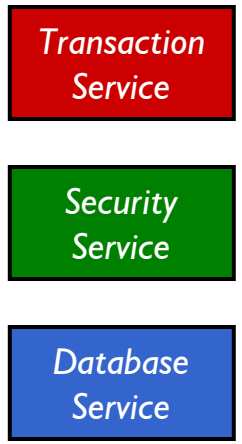
- Vários serviços que são necessários ou desejáveis em qualquer sistema distribuído
  - Comunicação síncrona e assíncrona (acoplamento fraco - messaging)
  - Autenticação, autorização, criptografia e segurança em geral
  - Redirecionamento transparente e balanceamento de cargas
  - Controle de transações e garantia de integridade das informações
  - Suporte a threads
  - Clustering (replicação), caching e gerenciamento de recursos
  - Acesso otimizado a conexões de banco de dados, soquetes, objetos
  - Integração com sistemas legados e de outros fabricantes
  - Gerenciamento remoto; geração de logs e estatísticas
- Desenvolver esses serviços exige amplos conhecimentos em áreas que pouco tem a ver com a aplicação
- Solução: usar middleware! Mas qual?

Tipos de middleware

- Temos uma aplicação com vários objetos distribuídos e um servidor que oferece serviços de middleware. E agora? Como acessar os serviços?
- Serviços podem ser acessados através de uma API
  - Sua aplicação precisará escrever o código para usar a API, por exemplo, JDBC para controlar acesso a bancos de dados, ou JTA para controlar transações, ou APIs de segurança: middleware explícito
- Serviços podem ser configurados através de interfaces ou arquivos de configuração externos ao código
  - Sua aplicação não precisa conter código algum de acesso a recursos externos. Você declara os serviços que precisa externamente: middleware implícito

Middleware explícito

- *Tem que ser implementado em cada método de cada objeto que deseja utilizar o serviço*



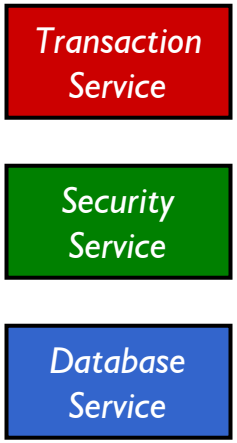
Esta é a única linha de lógica de negócio

```
public void transferir(Conta um, Conta dois, double valor) {
    // chama API para verificar segurança da requisição
    // chama API para iniciar a transação
    // chama API para carregar dados do banco
    // Subtraia o valor do saldo da conta um e some na conta dois
    // chama API para guardar dados no banco
    // chama API para fechar a transação
}
```

Fonte da ilustração: [4]

Middleware implícito

- *Escreva seu objeto para conter apenas lógica de negócio*
- *Declare os serviços de middleware que seu objeto precisa em um arquivo separado*
- *Rode uma ferramenta que pega o descritor e gera um objeto interceptador de requisições*




```
public void transferir(Conta um, Conta dois, double valor) {
   // Subtraia o valor do saldo da conta um e some na conta dois
}
```

Fonte da ilustração: [4]

Componentes de um EJB

- *Um EJB não é um único arquivo monolítico: são vários arquivos que trabalham juntos*
  - *Classe EJB*
  - *Interface Remote*
  - *Interface Home*
  - *Deployment descriptor*
  - *Empacotamento em JAR (EJB-JAR)*

Classe EJB

- *Contém a lógica de negócio*
- *É uma classe Java que obedece a uma interface definida e obedece certas regras*
- *Implementações diferem, dependendo do tipo*
  - *Session Beans - lógica relacionada a processos de negócio (computar preços, transferir fundos)*
  - *Entity Beans - lógica relacionada a dados (mudar o nome de um cliente, reduzir o saldo)*
  - *Message-driven Beans - lógica orientada a eventos (lógica assíncrona)*

Interfaces padrão

- A especificação defina algumas interfaces padrão que sua classe pode implementar
  - Forçam seu bean a expor certos métodos (definidos pelo modelo de componentes)
  - Container chama esses métodos para gerenciar seu bean e alertá-lo sobre eventos
- Todos implementam a interface *javax.ejb.EnterpriseBean*
  - implementa Serializable e não contém métodos
- Interfaces derivadas de EnterpriseBean
  - SessionBean
  - EntityBean
  - MessageDrivenBean

EJB Object

- *Enterprise Beans não são objetos remotos que podem ser usados de forma independente, como outros objetos RMI-IIOP*
  - *Um cliente (ou o skeleton) nunca chama o método diretamente em uma instância do bean: a chamada é interceptada pelo Container e delegada à instância*
  - *Ao interceptar requisições, o container pode automaticamente realizar middleware implicito*
- *O **EJBObject** é quem implementa a interface remota. Ele é o interceptador que chama o Bean*
  - *É objeto inteligente que sabe realizar a lógica intermediária que EJB container requer antes que uma chamada seja processada*
  - *É a "cola" entre o cliente e o bean: replica e expõe cada método de negócio do bean e delega todas as requisições*

Containers

- *Entidades abstratas*
- *Não são intermediários entre o cliente e o bean*
- *Podem interagir com beans chamando métodos de gerência quando necessário (métodos que só o container invoca) e avisam quando algum evento de middleware vai acontecer*
- *Conectam os clientes aos beans, oferecendo persistência, gerenciando o ciclo de vida, etc.*
- *Nem clientes nem beans explicitamente codificam para a API do container*
- *Container implicitamente gerencia o overhead de uma arquitetura de componentes distribuídos*

Interface Remote

- *Como as ferramentas sabem quais metodos de negocio devem ser replicados no EJBObject?*
- *Interface Remote é quem define todos os métodos de negócio expostos pelo bean*
- *Precisam ser derivadas de interface javax.ejb.EJBObject*
  - *Você declara mas não implementa os métodos de Remote (como faria com RMI). O container é quem gera o EJBObject que implementa a interface!*

RMI-IIOP versus EJB Objects

- Qualquer objeto que implementa javax.rmi.Remote é um objeto remoto e chamável de outra máquina virtual
  - Seus EJB Objects (gerados pelo Container) são objetos RMI-IIOP!
  - Interfaces Remote EJB são simplesmente interfaces Remote RMI-IIOP, só que precisam aderir à especificação EJB
- Regras de Remote
  - Todos os métodos provocam RemoteException
  - Interfaces Remote precisam suportar tipos de dados que podem ser passados via RMI-IIOP.
  - Tipos permitidos: primitivos, objetos serializáveis e objetos remotos RMI-IIOP

Instance pooling

- Gerência do ciclo de vida e atividade das instâncias de EJBs no servidor
- Servidor inteligentemente gerencia recursos (threads, conexões e objetos) para oferecer melhor desempenho, menos consumo de recursos
- Container dinamicamente instancia, destroi e reusa beans como for necessário
  - Se bean já existe na memória, container pode reutilizá-lo
  - Se bean está inativo, container pode destruí-lo
- Pool de beans pode ser bem menor que número de clientes conectados: "think time"

EJB Home

- Como clientes adquirem referências a objetos EJB?
  - Para obter uma referência a um objeto EJB, cliente usa uma fábrica de objetos EJB
  - Fábrica é responsável por operações do ciclo de vida de um objeto EJB (criar, destruir, encontrar)
  - Fábricas de EJBs são chamados de objetos EJBHome
- Objetos **EJBHome** servem para
  - Criar objetos EJB
  - Encontrar objetos EJB existentes
  - Remover objetos EJB
- São parte do container e gerados automaticamente
  - Implementam interface Home definida pelo programador

Interfaces locais

- Um problema das interfaces remotas é que criar objetos através dela é um processo lento
  1. Cliente chama um stub local
  2. Stub transforma os parâmetros em formato adequado à rede
  3. Stub passa os dados pela rede
  4. Esqueleto reconstrói os parâmetros
  5. Esqueleto chama o EJBObject
  6. EJBObject realiza operações de middleware como connection pooling, transações, segurança e serviços de ciclo de vida
  7. Depois que o EJBObject chama o Enterprise Bean, processo é repetido no sentido contrário
- Muito overhead! Em EJB 2.0 é possível chamar EJBs através de sua interface local
  1. Cliente chama objeto local
  2. Objeto EJB local realiza middleware
  3. Depois que o trabalho termina devolve o controle a quem chamou

Interfaces locais: conseqüências

- Interfaces locais são definidas para objetos EJBHome e para EJBObject (interfaces EJBLocal e EJBLocalHome)
- Benefícios
  - Pode-se escrever beans menores para realizar tarefas simples sem medo de problemas de performance
  - Uso típico: fachadas Session que acessam Entities
- São opcionais
  - Substituem ou complementam as interfaces remotas existentes
- Efeitos colaterais
  - Só funcionam ao chamar beans do mesmo processo (não dá para mudar o modelo de deployment sem alterar o código)
  - Parâmetros são passados por referência e não por valor: muda a semântica da aplicação!!!

Deployment descriptors

- Para informar ao container sobre suas necessidades de middleware, o bean provider deve declarar suas necessidades de middleware em um deployment descriptor (arquivo XML)
  - Informações de gerência e ciclo de vida (quem é o Home, o Remote, se é Session, Entity ou MDB)
  - Requisitos de persistencia para Entity Beans
  - Requerimentos de transações
  - Segurança (quem pode fazer o que com que beans, que metodos pode usar, etc.)
- Pode-se usar uma ferramenta (geralmente a ferramenta do próprio container) para gerar o arquivo

Vendor-specific files

- Cada servidor tem recursos adicionais que podem ser definidos em arquivos específicos
  - Como configurar load-balancing
  - clustering, pooling
  - monitoramento
  - mapeamento JNDI
- Geralmente são gerados por ferramentas no próprio servidor
- Podem também ser codificados à mão (jboss.xml)
- Devem ser incluidas no bean antes do deployment

EJB-JAR

- Arquivo JAR que contém tudo o que descrevemos
- Podem ser gerados de qualquer forma
  - ZIP, Ant, ferramentas dos containers
  - IDEs (NetBeans e deploytool)
- Uma vez feito o EJB-JAR, seu bean está pronto e é unidade implantável em application server
  - Ferramentas dos containers podem decomprimir, ler e extrair informações contidas no EJB-JAR
  - Depois, ferramentas realizam tarefas específicas do fabricante para gerar EJB Objects, EJB Home, importar seu bean no container, etc.
- Pode-se ter vários beans em um ejb-jar

helder@ibpinet.net

www.argonavis.com.br

Java 2 Enterprise Edition

Session Beans

Helder da Rocha
www.argonavis.com.br

Sumário

- *Esta apresentação aborda os principais conceitos sobre Session Beans*
- *Demonstração interativa será feita através dos exemplos do capítulo 10*
  - *Demonstração de Stateful Session Beans (Stateless Session Beans já foram vistos no capítulo 1)*

Session Beans

- São objetos de processo de negócio
  - Implementam lógica de negócio, algoritmos, workflow
  - Representam ações
- Uma das principais diferenças entre Session Beans e Entity Beans é o seu escopo de vida
  - Um Session Bean dura uma sessão (do cliente)
- Sessão
  - tempo que o browser está aberto
  - tempo que um outro bean usa o session bean
  - tempo que uma aplicação remota está aberta
- Objetos transientes
  - Não tem seu estado armazenado em meio persistente

Tipos de Session Beans

- Clientes travam um diálogo com um bean (o diálogo é a interação entre um cliente e um bean)
  - Consiste de uma ou mais chamadas entre cliente e bean
  - Dura um processo de negócios para o cliente
- Os dois tipos de session beans modelam tipos diferentes de diálogos
  - **Stateful** Session Beans modelam diálogos consistem de várias requisições onde certas requisições podem depender do estado de requisições anteriores
  - **Stateless** Session Beans modelam diálogos que consistem de apenas uma requisição

Stateless Session Beans

- *Como stateless session beans não mantém informação de estado do diálogo, todas as instâncias do mesmo bean são equivalentes e indistiguíveis*
  - *Não importa que chamou o bean no passado*
  - *Qualquer instância disponível de um session bean pode servir a qualquer cliente*
- *Session Beans podem ser guardados em um pool, reutilizados e passados de um cliente para outro em cada chamada*

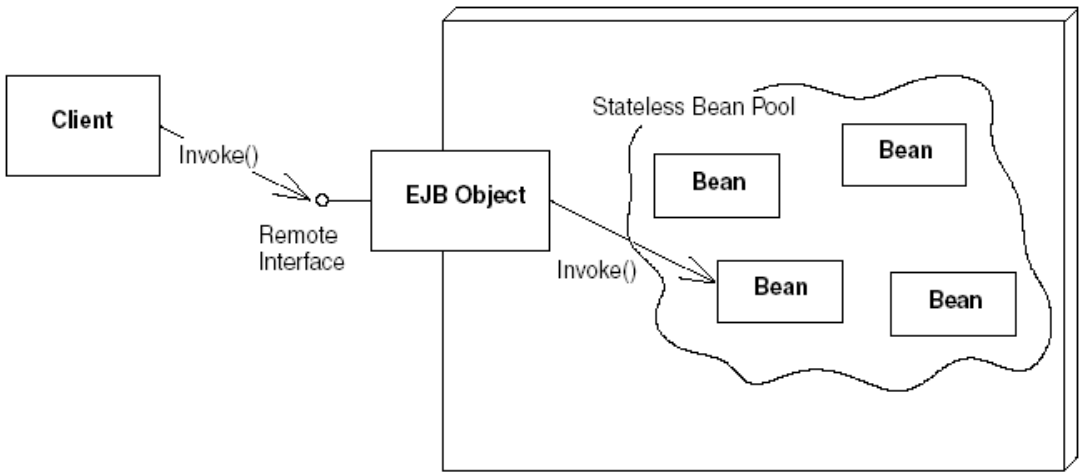


Figure 4.1 Stateless session bean pooling.

Fonte: [1]

Stateful Session Beans

- Quando um cliente chama um método de um bean, ele está iniciando um diálogo
  - O estado do diálogo deve ser mantido para a próxima requisição
  - Container não pode fazer o mesmo tipo de pooling que faz com stateless session beans
- Passivação e ativação
  - Solução para o problema de pooling
  - Dados do bean são armazenados em meio persistente durante a passivação e recuperados na ativação
  - Permite manter poucas instâncias no ar e vários clientes
  - Estratégia comum: LRU - Least Recently Used - Se container precisar de recursos, beans menos usados serão passivados. Logo que receberem uma requisição, serão reativados
- Objetos são serializados, portanto, fazem parte do estado do diálogo apenas objetos e variáveis não transientes

Passivação

- *Antes da passivação, método ejbPassivate() é chamado*

Fonte: [1]

Ativação

- *ejbActivate()* é chamado logo após a ativação

Fonte: [1]

Métodos da interface SessionBean

- *void setSessionContext(SessionContext ctx)*
  - *Associa bean com contexto da sessão*
  - *O contexto pode ser usado para obter referências para o interceptador e home do bean, se necessário*
- *void ejbCreate()*
  - *Stateless: só pode haver um, sem argumentos*
  - *Stateful: pode haver vários,com argumentos diferentes*
  - *Realiza a inicialização do bean*
- *void ejbRemove()*
  - *Chamado antes de liberar recursos e remover o bean da memória*
- *void ejbPassivate()*
  - *chamado antes de fazer o swap do bean para o disco*
  - *Stateless: não utiliza*
- *void ejbActivate()*
  - *chamado depois que estado do bean é recuperado do disco*
  - *Stateless: não utiliza*

Exemplos de Stateful Session Bean

- Nós já vimos exemplos de stateless session beans em exemplos anteriores
- Veja exemplo em cap10
  - mejb2: CountBean (comentado)
  - sun: CheckerBean e CartBean
- Para executar use o Ant (configure build.properties)
  - `ant jboss.deploy` (cria o JAR e copia para o deploy no JBoss)
  - `ant run.jboss.client` (executa um cliente de aplicação)
- Execução
  - Para demonstrar o efeito ejbActivate() e ejbPassivate(), o bean pool foi reduzido artificialmente (usando jboss.xml) para uma capacidade máxima de 2 beans
  - Como o cliente cria mais de dois beans, o container terá que passivar e depois ativar os beans (veja na saída da execução do JBoss quando cada método é chamado)

Interfaces Remote e Home

```
package examples;

public interface Count extends javax.ejb.EJBObject {

  public int count() throws java.rmi.RemoteException;

}
```

```
package examples;

public interface CountHome extends javax.ejb.EJBHome {

  Count create(int val)
        throws java.rmi.RemoteException, CreateException;

  Count create()
        throws java.rmi.RemoteException, CreateException;

}
```

Enterprise JavaBean

```
package examples;
public class CountBean implements javax.ejb.SessionBean {
    private SessionContext sessionContext;
    public int val;

    public int count() {
        System.out.println("count() chamado");
        return ++val;
    }
    public void ejbCreate(int val) throws CreateException {
        this.val = val;
        System.out.println("ejbCreate(val) chamado");
    }
    public void ejbCreate() throws CreateException {
        this.val = 0;
        System.out.println("ejbCreate() chamado");
    }
    public void ejbRemove() {
        System.out.println("ejbRemove() chamado");
    }
    public void ejbActivate() {
        System.out.println("ejbActivate() chamado");
    }
    public void ejbPassivate() {
        System.out.println("ejbPassivate() chamado");
    }
    public void setSessionContext(SessionContext ctx) {
         this.sessionContext = ctx;
    }
}
```

Deployment Descriptor

```
<!DOCTYPE ejb-jar PUBLIC
 "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
 "http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>CountEJB</ejb-name>
      <home>examples.CountHome</home>
      <remote>examples.Count</remote>
      <ejb-class>examples.CountBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Ciclo de vida: Stateless Session Bean

Fonte: [1]

Ciclo de vida: Stateful Session Bean

Fonte: [1]

Exercício

- 1. Comparação Stateful-Stateless
  - Remova o método create() com argumentos do exemplo do CountBean (deixando apenas o sem argumentos). Adapte o cliente para que ele use apenas create() e não create(valor)
  - Execute a aplicação
  - Mude, no deployment descriptor, o bean para Stateless e rode a aplicação novamente.
- 2. Veja cap10/exercicio
  - A partir dos arquivos NomesBase.java e Nomes.java (este uma interface Remote), construa um Stateful Session Bean (Home, Bean, ejb-jar.xml) que funcione com o cliente NomesClient fornecido.
  - Utilize o jboss.xml fornecido

Fontes

[1] *Ed Roman, Mastering EJB 2, 2002, Capítulo 4.*
[2] *Dale Green. Session Beans. J2EE Tutorial, Sun, (veja CD)*

helder@ibpinet.net

www.argonavis.com.br

Entity Beans com BMP

Helder da Rocha
www.argonavis.com.br

Introdução

- *Entity beans são objetos persistentes que podem ser armazenados em meio persistente de armazenamento*
  - *Pode-se fazer modelagem de dados como Entity Beans*
- *Tópicos deste módulo*
  - *Conceitos básicos de persistência*
  - *Definição de entity beans*
  - *Recursos*
  - *Conceitos de programação*
  - *Típos de entity beans*
  - *Exemplos de entity beans usando Bean-Managed Persistence*

Persistência

- *Uma maneira de armazenar o estado de objetos Java é através da serialização*
  - *Vantagem: facilidade de leitura e gravação: objeto pode ser enviado pela rede, armazenado em um banco ou árvore JNDI*
  - *Desvantagem: inviabilidade de realização de pesquisa sobre campos dos objetos: qualquer pesquisa necessitaria a desserialização de cada objeto para obter os campos pesquisados*
  - *Conclusão: para persistência de grande quantidade de dados que precisam de expor seus dados como atributos de pesquisa, serialização não é viável (solução seria serialização em XML!)*
- *Soluções alternativas*
  - *Bancos de dados relacionais: oferecem recursos poderosos e eficientes de busca, mas, não são orientados a objeto (SQLJ, JDBC)*
  - *Bancos de dados orientados a objeto*
  - *Mapeamento objeto-relacional (O/R Mapping)*

Object-Relational Mapping

- *Em vez de serializar os objetos, decompomos cada um em suas partes e guardamos suas partes no banco*

Fonte: [1]

O que é um Entity Bean?

- Session beans modelam a lógica relacionada a tarefas realizadas pela aplicação
- Entity beans modelam entidades
- São componentes que representam dados
  - Session beans podem guardar informações em bancos de dados mas não "representam" dados.
  - Entity beans são os dados
- Permitem que se manipule com objetos ignorando a forma de armazenamento
  - A especificação não define nenhum tipo obrigatório
- Duas partes
  - Entity bean instance: os dados na memória (representa uma instância da classe do Entity Bean)
  - Entity bean data: os dados fisicamente armazenados no banco

Partes de um entity bean

- Como qualquer EJB, um Entity Bean possui
  - Uma interface Home ou LocalHome
  - Uma interface Remote ou Local
  - Um deployment descriptor
  - Uma classe EJB
- As diferenças são
  - A classe do EntityBean está mapeada à uma definição de entidade de um esquema de banco de dados (uma tabela, por exemplo)
  - Há métodos especiais que lidam com a sincronização do EntityBean com o banco de dados
  - Um entity bean tem uma classe Primary Key que permite identificar univocamente sua instância

Entity Beans sobrevivem

- Entity Beans são objetos persistentes
  - Sobrevivem à quedas do servidor ou banco de dados
  - Por serem apenas representação de dados de um meio de armazenamento, o bean pode ser reconstruido lendo seus dados do banco.
- Ciclo de vida muito longo
  - Existem enquanto existir a tabela ou entidade que representam seus dados no banco
  - Muitas vezes já existem antes da construção da aplicação

Sincronismo

- *Uma instância de um entity bean é uma visão do banco de dados*
- *A instância está sempre sincronizada com os dados do banco. Container possui mecanismo que faz a atualização após cada alteração*
  - *Todas as classes implementam métodos que realizam essa atualização*
  - *Container chama os métodos automaticamente*
- *void ejbLoad()*
  - *lê os dados do banco para dentro do entity bean*
- *void ejbStore()*
  - *grava os dados do entity bean no banco*

ejbLoad() e ejbStore()

Fonte: [1]

Uso e reuso de Entity Beans

- *Instâncias diferentes de entity beans podem representar os mesmos dados*
  - *Transações permitem isolamento necessário para evitar corrupção*
- *Mesmas instâncias podem representar dados diferentes*

Fonte: [1]

Duas formas de persistência

- Entity beans podem ser associados ao meio de armazenamento de duas formas
- **BMP** - Bean Managed Persistence
  - O bean deve usar uma API (JDBC, SQLJ) para fazer as chamadas de UPDATE no ejbStore(), INSERT no ejbCreate(), SELECT no ejbLoad(), etc. (caso use banco relacional)
- **CMP** - Container Managed Persistence
  - O container faz um mapeamento dos objetos a entidades em um banco
  - Métodos de ciclo de vida, campos de dados e métodos de negócio são gerados automaticamente pelo container

Criação e remoção de Entity Beans

- Como Entity Beans são os dados, criar um bean é inserir dados em um banco; remover um bean é apagar esses dados
- Entity Beans são criados através do EJBObject (proxy) via sua interface Home.
- Se você tiver o seguinte ejbCreate() no seu bean

```
public AccountPK ejbCreate(String id, String owner) {}
```

- Você deve ter o seguinte create() no seu Home

```
public Account create(String id, String owner)  {}
```

- Observe que os tipos retornados são diferentes
  - O bean retorna uma chave primária para o container (Home) que a utiliza para achar o bean para o cliente (que chamou create())
- Para destruir um bean, é preciso chamar remove() em Home
  - ejbRemove() deve localizar a chave primária do bean atual para removê-lo através o método getPrimaryKey() do contexto.

ejbCreate() e ejbRemove()

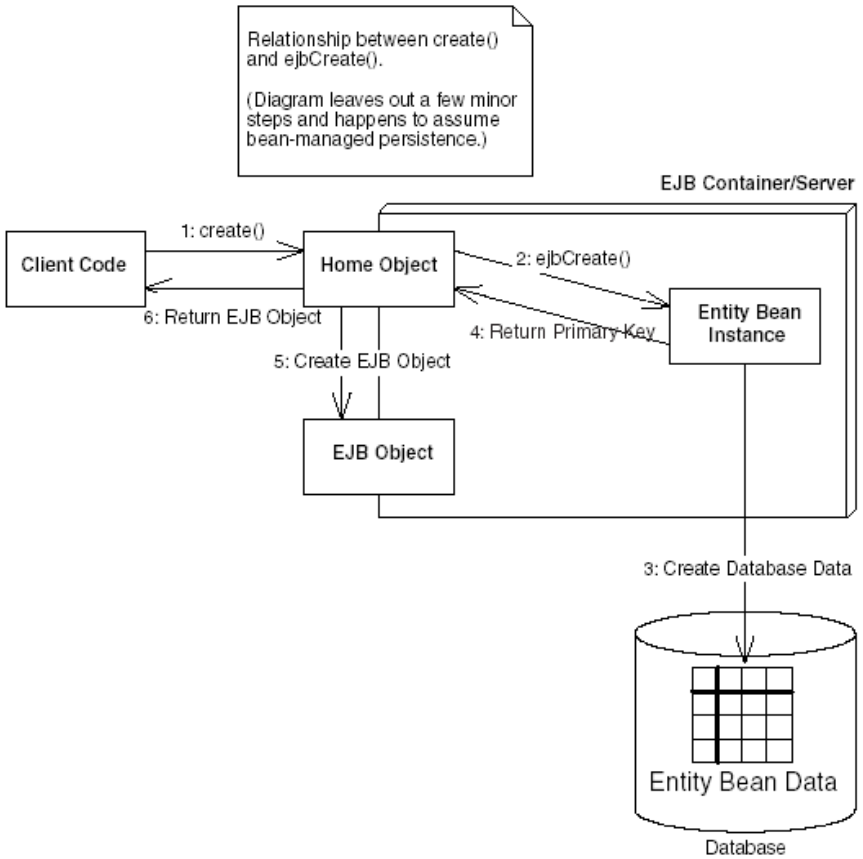


Figure 5.6 Creating an entity bean and EJB object.

Fonte: [1]

Entity Beans podem ser localizados

- Entity Beans nem sempre precisam ser criados. Podem já existir no banco
  - Neste caso, devem ser localizados
  - Todo bean pode ser localizado através de sua chave primária
  - Métodos find() adicionais podem ser criados para localizar beans de acordo com outros critérios
- Finder methods
  - No home: findByXXX()
  - No EJB: ejbFindByXXX()
  - Deve existir pelo menos o findByPrimaryKey()
  - Método create() é opcional

Entity Beans podem ser modificados sem EJB

- Não é preciso usar EJB para modificar o conteúdo de Entity Beans
  - Como Entity Beans representam dados externos, se esses dados puderem ser alterados externamente, sua mudança será refletida no Entity Bean
  - Riscos: cache de entity beans pode causar inconsistência

Fonte: [1]

Entity Contexts

- Todo bean possui um objeto de contexto
- No caso do Entity Bean é o javax.ejb.EntityContext
- Através dele pode-se chamar os métodos
  - getEJBObject(): retorna uma referência para o EJBObject (interceptador, que implementa Remote)
  - getEJBLocalObject(): retorna uma referência local para o interceptador
  - getPrimaryKey(): retorna a chave primária associada com a entidade atual.
- getPrimaryKey() é essencial em ejbLoad() e ejbRemove() para descobrir qual a instância que será carregada ou removida, respectivamente

Métodos de um Entity Bean (1)

- *void setEntityContext(EntityContext ctx)*
  - *Obtém o contexto do Entity Bean. Pode ser usado para obter a chave primária da instância atual*
- *PK ejbFindXXX(...)*
  - *Deve haver pelo menos ejbFindByPrimaryKey(), que devolve a própria PrimaryKey*
  - *Cada método ejbFind() deve devolver ou a chave primária ou uma Collection de PKs (pode ser Set)*
- *tipo ejbHomeXXX(...)*
  - *Métodos de Home que não se referem a nenhuma instância em particular. XXX é o nome do método na interface Home.*
- *PK ejbCreate(...)*
  - *Não precisa existir um ejbCreate()*
  - *Deve retornar PK ou Collection de PKs (pode ser Set)*
- *void ejbPostCreate(...)*
  - *Deve haver um para cada ejbCreate() com mesmos argumentos*

Métodos de um Entity Bean (2)

- *void ejbActivate()*
  - *Chamado logo após a ativação do bean*
- *void ejbLoad()*
  - *Chamado logo depois que o bean for povoado com os dados oriundos do banco (antes de qualquer alteração nos dados do bean)*
- *void ejbStore()*
  - *Chamado antes que o bean seja armazenado no banco (depois de qualquer alteração nos dados do bean)*
- *void ejbPassivate()*
  - *Chamado antes da passivação do bean*
- *void ejbRemove()*
  - *Chamado antes que os dados sejam removidos do banco*
- *void unsetEntityContext()*
  - *destrói instância (será recolhida pelo garbage collector). Isto destrói o objeto, mas não o entity bean (que são os dados)*

Métodos da interface Home

- Remote *create*(...)
  - Retorna uma referência para o bean criado
  - Pode retornar uma Collection (ou Set) de referências
  - Opcional
- Remote *findXxx*(...)
  - Retorna uma referência para bean existente
  - Pode retornar uma Collection (ou Set) de referências
  - Deve haver pelo menos findByPrimaryKey()
- tipo *xxx*(...)
  - Método chamado na interface Home não se refere a nunhuma instância particular
  - Deve ter um correspondente *ejbHomeXxx()* no bean com mesmos argumentos e tipo de retorno

Resumo: relacionamento Home-Bean

- *Os métodos de EJBHome à esquerda delegam chamadas aos métodos de EntityBean à direita*
  - *Ou seja, se você tiver os métodos à esquerda no seu Home, deve ter os métodos à direita no seu Bean*

Interface Home	**EJB**
Objeto findByPrimaryKey(ObjetoPK)	ObjetoPK ejbFindByPrimaryKey(ObjetoPK)
Objeto findByParams(Par1, Par2)	ObjetoPK ejbFindByParams(Par1, Par2)
Collection findByParams(Par1, Par2)	Collection ejbFindByParams(Par1, Par2)
Objeto create(Par1, Par2)	ObjetoPK ejbCreate(Par1, Par2) void ejbPostCreate(Par1, Par2)
Tipo operacao(Par1, Par2)	Tipo ejbHomeOperacao(Par1, Par2)

- *Chave*
  - *Objeto: interface do objeto remoto*
  - *ObjetoPK: chave primária do objeto remoto*
  - *Par1, Par2 e Tipo: tipos primitivos, remotos ou serializáveis*

Aplicação exemplo

- *Veja cap11/mejb2*
  - *Código-fonte em diretório src/*
  - *Deployment-descriptor em deployment/*

Fonte: [1]

Deployment Descriptor

```
<ejb-jar>
    <enterprise-beans>
        <entity>
            <ejb-name>Account</ejb-name>
            <home>examples.AccountHome</home>
            <remote>examples.Account</remote>
            <local-home>examples.AccountLocalHome</local-home>
            <local>examples.AccountLocal</local>
            <ejb-class>examples.AccountBean</ejb-class>
            <persistence-type>Bean</persistence-type>
            <prim-key-class>examples.AccountPK</prim-key-class>
            <reentrant>False</reentrant>
            <resource-ref>
                <res-ref-name>jdbc/ejbPool</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
            </resource-ref>
        </entity>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>Account</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

Fonte de dados onde bean está armazenado

Execução

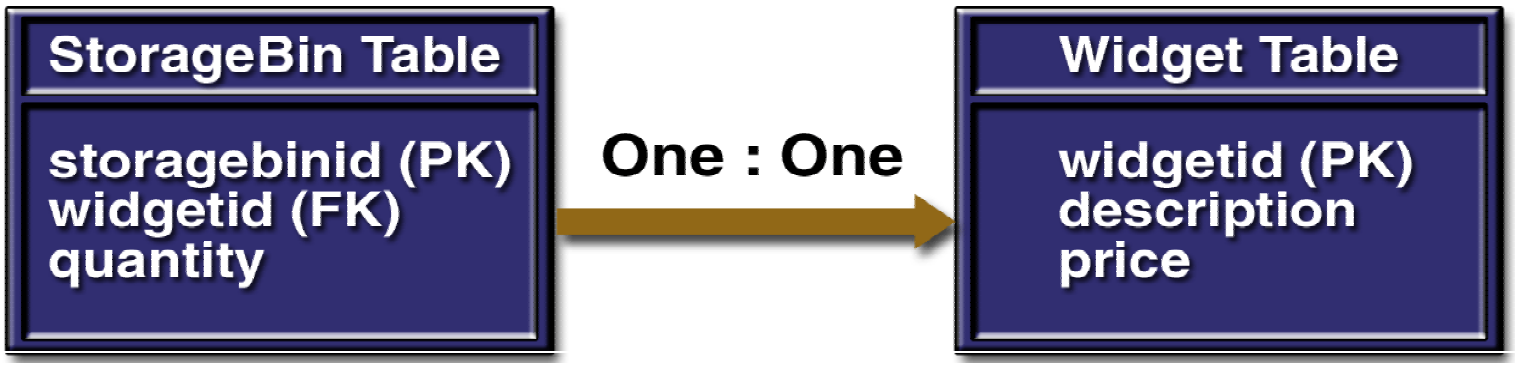
- *Para executar a aplicação exemplo, use o Ant no diretório cap11/mejb2*
  - *Configure o build.properties*
- *Etapas*
  - *Povoar banco de dados:* `ant db.insert`
  - *Deployment:* `ant jboss.deploy`
  - *Rodar o cliente:* `ant run.jboss.client`
- *Manhenha a janela do JBoss visível para ver simultaneamente as chamadas no cliente e servidor*

Ciclo de vida

Fonte: [1]

Exemplos do tutorial da Sun

- *4 exemplos (foram modificados para rodar no JBoss)*
  - *Veja em cap11/sun/relationships/*
  - *Montar os bancos de dados:* `ant db.insert`
  - *Para fazer o deploy:* `ant jboss.deploy`
- *Relacionamento BMP 1:1*
  - *Para rodar:* `ant run.one2one.client`

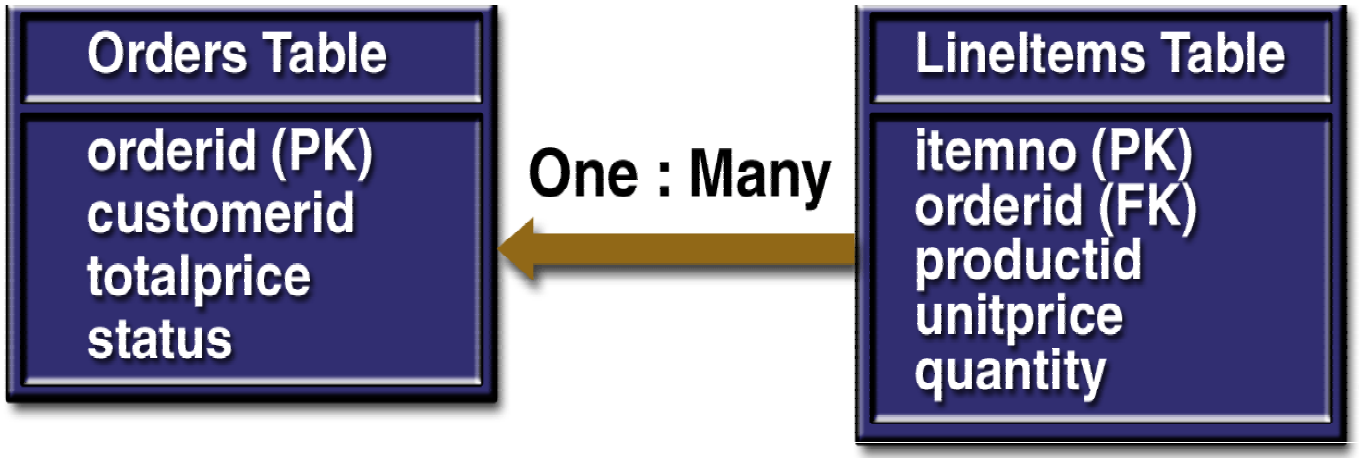


src/j2eetut/storagebin

Os relacionamentos são implementados nas tabelas e no código: veja os scripts SQL, os constraints e também o código das interfaces

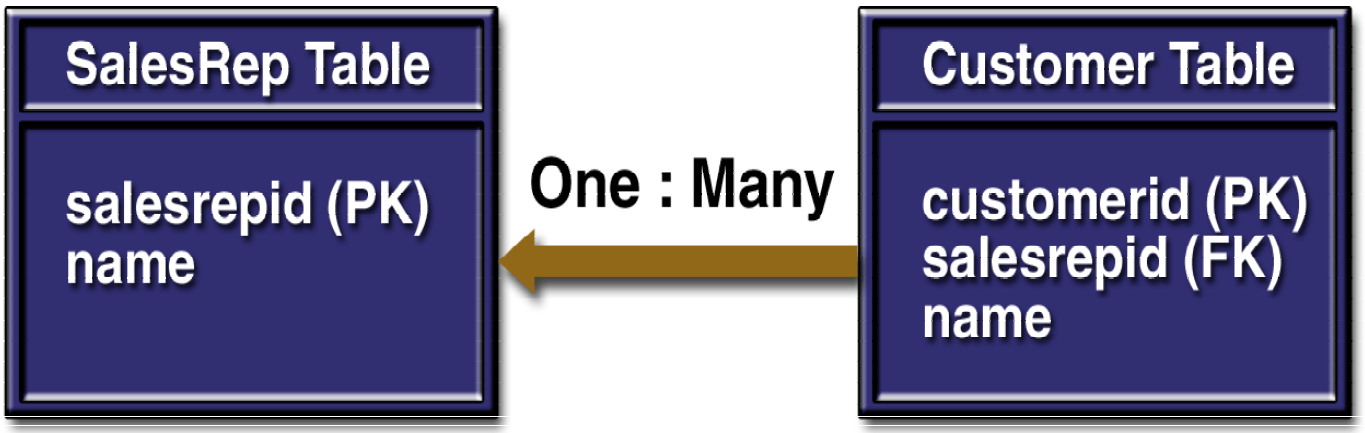
Exemplos Tutorial da Sun (2)

- *Relacionamentos BMP 1:* (dois exemplos)*
  - *Para rodar:* `ant run.one2many.helper.client`



src/j2eetut/order

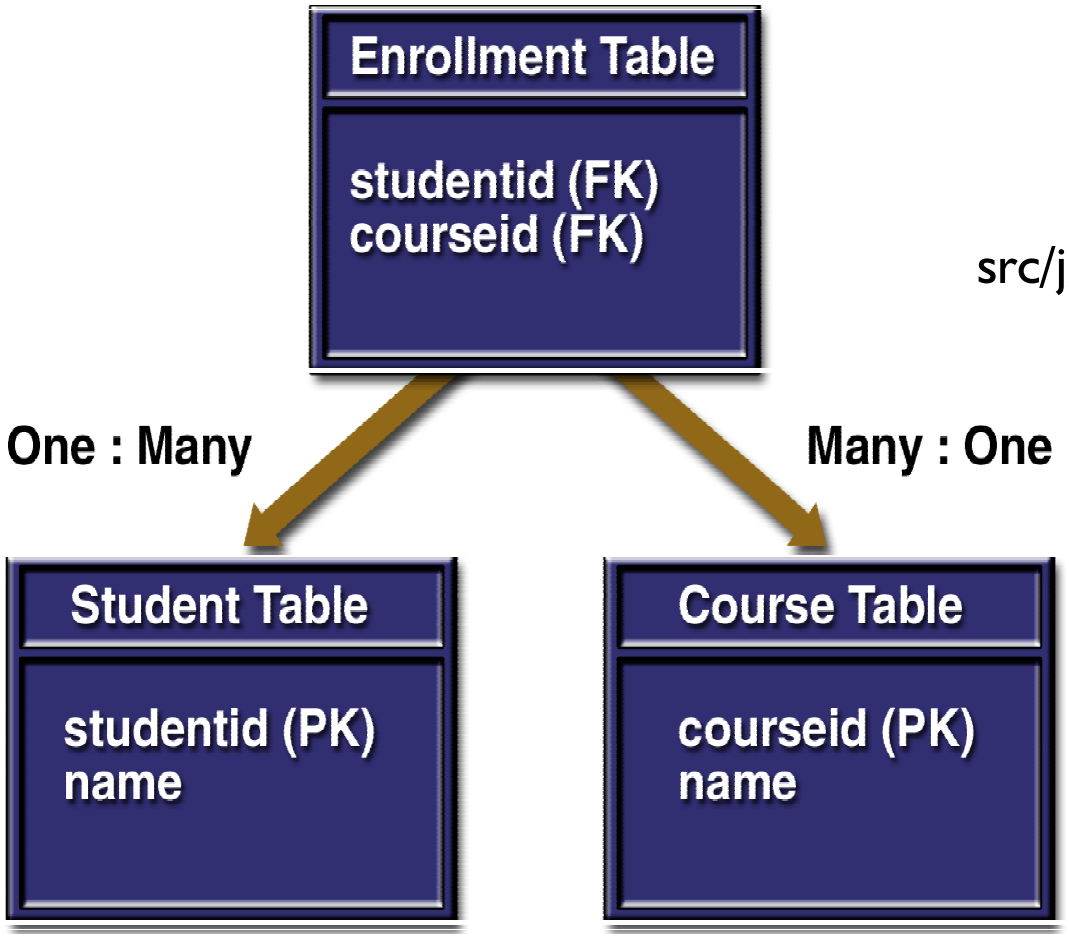
  - *Para rodar:* `ant run.one2many.helper.entity`



src/j2eetut/salesrep

Exemplos Tutorial da Sun (3)

- *Relacionamentos *:**
  - *Para rodar:* `ant run.many2many.client`



src/j2eetut/enroller

Exercício

- 1. Implemente um EntityBean que represente a seguinte tabela
  - Crie, no pacote exercicio, as classes Veiculo, PlacaPK, VeiculoHome, VeiculoBean e o ejb-jar.xml (já estão prontos VeiculoClient), o jboss.xml e build.xml

```
create table veiculos (
    placa varchar(16) primary key,
    marca varchar(32),
    entrada numeric(18),
    servicos numeric(18)
);
```

  - A primary key PlacaPK deve conter apenas a placa
  - Defina os seguintes finders na sua interface Home: findByPrimaryKey(placa), findByMarca(marca), getTotalVeiculos()
  - Faça o exercício dentro do diretório cap11/exercicio1/src/
- 2. No diretório capitulo2 há um esqueleto de aplicação. Termine a aplicação definindo o conteúdo dos métodos do bean e execute-a.

Fontes

[1] Ed Roman, Mastering EJB 2, 2002. Fonte da maior parte dos diagramas usados e do exemplo (modificado).

[2] Dale Green. Bean-Managed Persistence Examples. Sun J2EE Tutorial, 2002. Fonte dos exemplos (modificados para rodar no JBoss)

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)

© 1999-2003, Helder da Rocha

Message-driven beans

Helder da Rocha
www.argonavis.com.br

Objetivos

- *Este capítulo é uma breve introdução aos Message-Driven Beans (MDB).*
- *MDBs são consumidores de mensagens JMS*
  - *Já sabemos como criar produtores e consumidores JMS*
  - *Neste capítulo veremos apenas como transformá-los em enterprise Java Beans*
- *Para quem deseja se aprofundar mais no assunto, veja o capítulo "Message Driven Beans" no J2EE Tutorial eas implementações de MDBs no Duke's bank.*

O que é um Message-Driven Bean

- Bean guiado por mensagens, ou eventos
  - Um event-handler distribuído em forma de EJB
- MDBs consomem mensagens enviadas a filas e canais JMS
  - Não é possível enviar uma mensagem diretamente a um message-driven bean (envia-se mensagens a um canal que o bean escuta)
  - Acoplamento entre clientes e MDB resume-se à conhecimento de canal comum de comunicações
- Um message-driven bean é um EJB mas
  - Não possui interfaces Home ou Remote
  - Possui apenas um método que recebe qualquer tipo de mensagem
  - Não devolve tipos ou exceções
  - Não possui estado (são stateless)
  - Podem ser ouvintes de uma fila ou assinantes duráveis ou não-duráveis de um canal (Topic) JMS

Desenvolvendo MDB

- *Message-driven beans implementam duas interfaces*

```
public interface javax.jms.MessageListener
public interface javax.ejb.MessageDrivenBean
                    extends javax.ejb.EnterpriseBean
```

- *Métodos de MessageDriveBean*
  - *ejbRemove(): chamado pelo container quando o message-driven bean está sendo removido do pool*
  - *ejbCreate(): chamado quando um novo message-driven bean é instanciado*
  - *setMessageDrivenContext(MessageDrivenContext context): chamado antes do ejbCreate() quando o bean está sendo adicionado ao pool*
- *Método de MessageListener*
  - *onMessage(Message m): chamado cada vez que uma mensagem é enviada para o canal do bean (se o bean estiver ativado).*

Exemplo: implementação de um MDB

```
import javax.ejb.*;
import javax.jms.*;

public class LogBean implements MessageDrivenBean, MessageListener {

    protected MessageDrivenContext ctx;
    public void setMessageDrivenContext(MessageDrivenContext ctx) {
        this.ctx = ctx;
    }
    public void ejbCreate() {}
    public void ejbRemove() {}

    public void onMessage(Message msg) {
        TextMessage tm = (TextMessage) msg;
        try {
            String text = tm.getText();
            System.err.println("Received new message : " + text);
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

Métodos da interface MessageDrivenBean

Método da interface MessageListener

MDB não pode provocar exceção alguma

Este bean simplesmente imprime o texto contido na mensagem recebida

Ciclo de Vida

- Ciclo de vida muito simples
  - Apenas dois estados

Deployment Descriptor

```
<ejb-jar>
  <enterprise-beans>
    <message-driven>

      <ejb-name>LogEJB</ejb-name>
      <ejb-class>examples.LogBean</ejb-class>
      <transaction-type>Container</transaction-type>

      <message-driven-destination>
        <destination-type>javax.jms.Topic</destination-type>
        <subscription-durability>NonDurable</subscription-durability>
      </message-driven-destination>

      <resource-ref>
         <res-ref-name>jms/TopicFactory</res-ref-name>
         <res-type>javax.jms.TopicConnectionFactory</res-type>
         <res-auth>Container</res-auth>
      </resource-ref>

    </message-driven>
  </enterprise-beans>
</ejb-jar>
```

Canal (apenas a classe e tipo)

Fábrica de filas e canais

Vendor-specific

- *Os nomes JNDI dos recursos que serão acessados globalmente (nome da fila ou canal, por exemplo), são definidos na configuração do fabricante*
- *No JBoss:*

```
<jboss>
    <enterprise-beans>
        <message-driven>
            <ejb-name>LogEJB</ejb-name>
            <destination-jndi-name>
              topic/testTopic
            </destination-jndi-name>
            <resource-ref>
                <res-ref-name>jms/TopicFactory</res-ref-name>
                <jndi-name>ConnectionFactory</jndi-name>
            </resource-ref>
        </message-driven>
    </enterprise-beans>
</jboss>
```

Nome JNDI global do destino compartilhado pelo bean e pelos clientes

Referência do ENC java:comp/env

Nome JNDI global default da fábrica de destinos (fila ou canal) no JBoss

Cliente é um cliente JMS comum

- *Pode ser outro bean, cliente Web ou standalone*

```
public class Client {

    public static void main(String[] args) throws Exception {

        Context ctx = new InitialContext(System.getProperties());
        TopicConnectionFactory factory =
             (TopicConnectionFactory) ctx.lookup("ConnectionFactory");
        TopicConnection connection =
             factory.createTopicConnection();
        TopicSession session =
             connection.createTopicSession(
                  false, Session.AUTO_ACKNOWLEDGE);

        Topic topic = (Topic) ctx.lookup("topic/testTopic");
        TopicPublisher publisher = session.createPublisher(topic);
        TextMessage msg = session.createTextMessage();
        msg.setText("This is a test message.");
        publisher.publish(msg);
    }
}
```

Cliente externo (fora do ENC) usa nomes JNDI globais

Execução dos exemplos

- Diretório cap12/mejb2/
- Para montar e instalar a aplicação, use

```
> ant jboss.deploy
```

- Uma vez instalada, rode o cliente:

```
> ant run.jboss.client
```

- A execução não deve causar nenhuma reação no cliente, já que foi assíncrona "one-way".
- No servidor, aparece a mensagem enviada:

```
[STDERR] Received new message : This is a test message.
```

- Veja também o exemplo do J2EE Tutorial (usando Queue) no subdiretório cap12/sun/

Exercício

- *1. Transforme os consumidores de mensagens JMS usados como exemplo no capítulo 4 em message-driven beans*
  - *a) Chame os beans a partir de um session bean que recebe a mensagem e o número de vezes em que ela é chamada de um cliente standalone ou Web.*

Fontes

[1] *Ed Roman*. *Mastering EJB 2. Chapter 8 - Introduction to Message Driven beans*.

[2] *J2EE Tutorial*. *Message-driven Beans*.

helder@ibpinet.net

www.argonavis.com.br

Entity beans com CMP

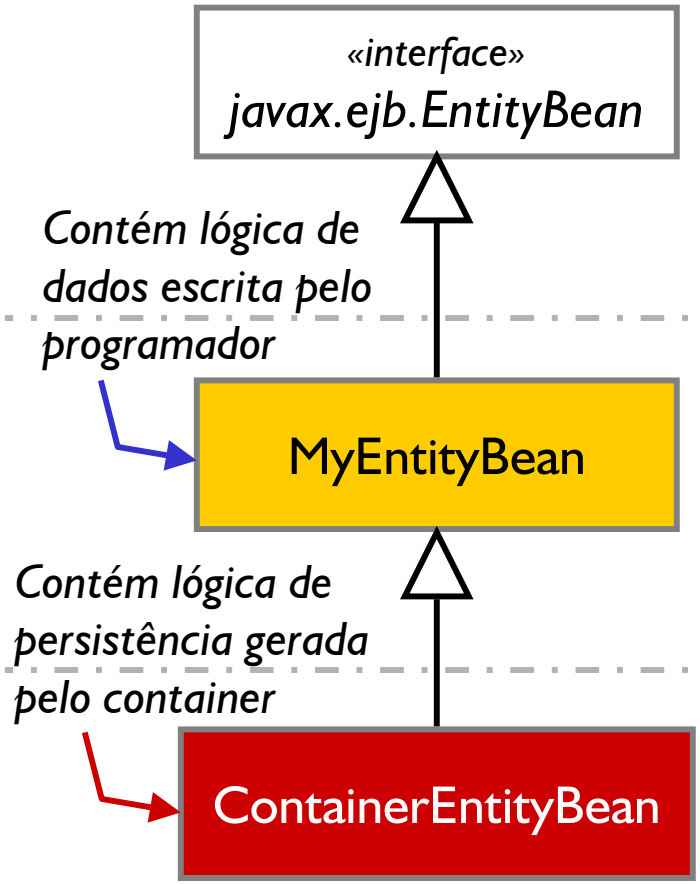
Helder da Rocha
www.argonavis.com.br

Objetivos

- *Este módulo é uma continuação do módulo 11 (Entity Beans).*
- *Desta vez veremos como criar beans de entidade persistentes sem escrever uma linha sequer de SQL e sem usar JDBC*
  - *O resultado: beans muito mais simples contendo apenas a lógica de negócio!*
- *CMP ainda é bastante limitado (não dá para substituir totalmente BMP ainda)*
- *Não abordaremos conceitos avançados de relacionamentos (apenas os fundamentos de CMR)*
  - *Para mais informações, consulte as referências!*

Características de EJB CMP

- *Com container-managed persistence o programador do bean não implementa lógica de persistência*
  - *O bean não precisa usar APIs como JDBC*
  - *O container se encarrega de criar as tabelas (no deployment), criar os beans e os dados, removê-los e alterá-los*
- *O container gera automaticamente o JDBC ao estender* a classe do bean criada pelo programador*
  - *O entity bean verdadeiro é o que foi gerado: a combinação da classe implementada pelo programador e a gerada pelo container*



** Na verdade, o container tem a liberdade de utilizar outros mecanismos (delegação, por exemplo), desde que produzam o mesmo resultado*

Entity Beans CMP não tem ...

- ... campos de dados declarados
  - São implementados na subclasse* (criada pelo container)
  - Programador declara seus campos no deployment descriptor (abstract persistence schema) usando <cmp-field>
- ... implementação de métodos de acesso
  - Programador declara todos os seus métodos de acesso (get/set) como abstract
  - Os nomes dos métodos devem ser compatíveis com os nomes dos campos declarados:

    Se há `getCoisa()` no bean, e no deployment descriptor deve haver:

    `<cmp-field><field-name>coisa</field-name></cmp-field>`
  - Subclasse* gerada pelo container implementa os métodos (o container pode usar BMP, por exemplo)

* ou no mecanismo alternativo implementado pelo container

EJB-QL

- *Como não há como fazer queries JDBC como entity beans CMP, há uma linguagem para fazer queries nos próprios beans: Enterprise JavaBean Query Language (EJB-QL)*
- *EJB-QL é uma linguagem com sintaxe semelhante a SQL para pesquisar entity beans. Possui*
  - *cláusula SELECT*
  - *cláusula FROM*
  - *cláusula WHERE, opcional*
- *EJB-QL é colocado nos deployment descríptors no contexto da declaração do método. Exemplo*

```
SELECT OBJECT(obj)
    FROM ProductBean obj
    WHERE obj.name = ?1
```

Métodos da interface EntityBean (I)

- *setEntityContext() / unsetEntityContext()*
  - *Mesmo comportamento de entity beans com BMP*
- *ejbFindXXX(...)*
  - *Não devem ser implementados em CMP*
  - *Use EJB-QL para definir ações de pesquisa para cada método no DD*
- *ejbSelectXXX(...)*
  - *Usados para fazer queries genéricos para o bean*
  - *É método de negócio mas não pode ser chamado pelo cliente*
  - *Declarados como abstract com lógica descrita no DD em EJB-QL*
- *ejbHomeXXX(...)*
  - *Usado para definir métodos que não são específicos a uma instância*
  - *Deve chamar um ejbSelect() para realizar as operações*
- *ejbCreate(...) / ejbPostCreate(...)*
  - *Chama os métodos de acesso abstratos inicializando-os com os dados que serão usados pelo container para criar novo registro*

Métodos (2)

- *ejbActivate() / ejbPassivate()*
  - *Mesmo comportamento que BMP*
- *ejbLoad()*
  - *Usado para carregar dados do banco para o bean*
  - *Pode ser vazio ou conter rotinas para transformar dados recebidos*
- *ejbStore()*
  - *Usado para atualizar o banco com alterações no bean*
  - *Pode ser vazio ou conter rotinas p/ transformar dados antes do envio*
- *ejbRemove()*
  - *chamado por home.remove(), remove os dados do banco (não remove a instância - que pode ser reutilizada)*
  - *Pode ser vazio ou conter código para ser executado antes da remoção dos dados*

Diferenças BMP-CMP

Diferença	Container-Managed Persistence	Bean-Managed Persistence
Definição da classe	Abstrata	Concreta
Chamadas de acesso ao banco de dados	Gerada pelas ferramentas no deployment	Codificada pelo programador
Estado persistente	Representadas como campos persistentes virtuais	Codificadas como variáveis de instância
Métodos de acesso a campos persistentes e relacionamentos	Obrigatórios (abstract)	Não há
Método findByPrimaryKey	Gerado pelo container	Codificado pelo programador
Métodos finder customizados	Gerados pelo container mas programador deve escrever EJB-QL	Codificado pelo programador
Métodos select	Gerados pelo container	Não há
Valor de retorno de ejbCreate()	Deve ser null	Deve ser a chave primária

Fonte: J2EE Tutorial

Exemplo CMP

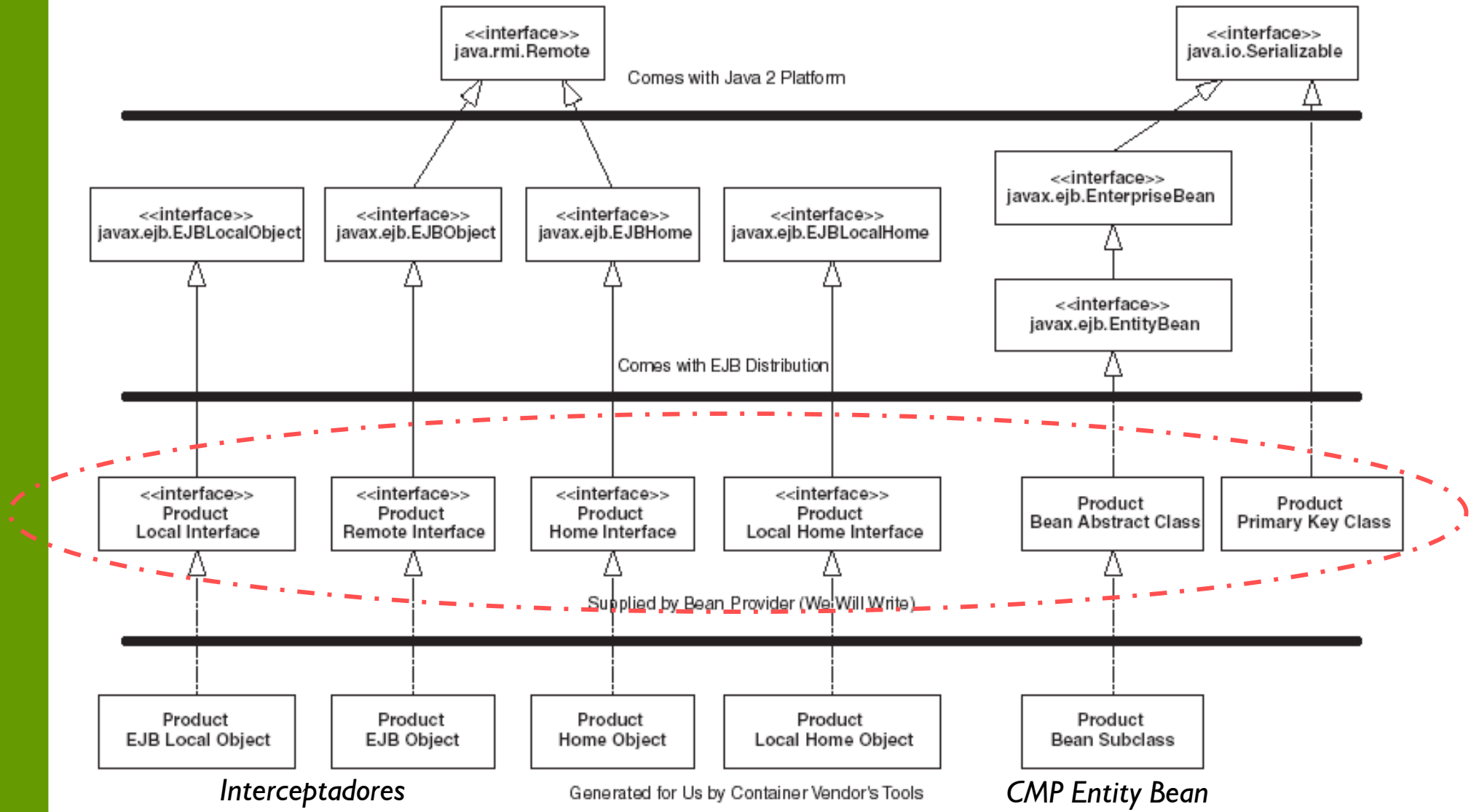


Figure 7.3 The object model for our product line.

Exemplo: Como executar

- Diretório *cap13/mejb/*
- Configuração
  - *Configure o arquivo build.properties com seu ambiente*
  - *Inicie o JBoss e o servidor de banco de dados Cloudscape*
- Inicialização

```
> ant drop.table
> ant clean
```

- Deployment

```
> ant jboss.deploy
```
(observe as mensagens de criação das tabelas no JBoss)

- Execução do cliente

```
> ant run jboss.client
> ant select.all
```
(faz SELECT na tabela)

Componentes do EJB-JAR

- Interfaces Home e Remote e classe Primary Key
  - Idênticas às interfaces e classe Primary Key de um entity bean equivalente que usa BMP
  - PK não pode conter campos de relacionamento (CMR)
- Classe Enterprise bean (menor que classe BMP)
  - Métodos ejbLoad(), ejbStore() não possuem código de persistência
  - Classe é abstrata; métodos de acesso são abstratos; métodos ejbSelectXXX() são abstratos
  - Métodos de negócio contém apenas lógica de negócio
  - Métodos finder não são declarados
- Deployment descriptor (maior que versão BMP)
  - Definição da lógica de métodos e relacionamentos
- Arquivos de configuração do fabricante (JBoss)
  - Mapeamento de tipos, mapeamento de esquema

Enterprise Bean

```
public abstract class ProductBean implements EntityBean {
  protected EntityContext ctx;

  public abstract String getName();
  public abstract void setName(String name);
  public abstract String getProductID();
  public abstract void setProductID(String productID);
    (...)
  public void ejbStore() {}
  public void ejbLoad() {}
  public void ejbPassivate() {}
  public void ejbRemove() {}
  public void ejbActivate() {}
  public void setEntityContext(EntityContext c) {ctx = c;}
  public void unsetEntityContext() {ctx = null;}
  public void ejbPostCreate(String productID, String name,...) {}
  public void ejbCreate(String productID, String name, ...) {
    setProductID(productID);
    setName(name);
    (...)
  }
}
```

Métodos de negócio abstratos (lógica no deployment descriptor)

ejbLoad *e* ***ejbStore*** *são implementados pelo container*

ejbFindBy *não precisam ser declarados (lógica no deployment descriptor)*

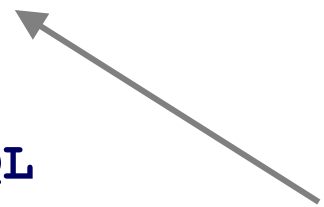
ejbCreate *chama métodos abstratos para passar dados recebidos*

Deployment Descriptor (1)

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>ProductEJB</ejb-name>
      <home>examples.ProductHome</home>
      <remote>examples.Product</remote>
      <ejb-class>examples.ProductBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>examples.ProductPK</prim-key-class>

      <reentrant>False</reentrant>

      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>
         ProductBean
      </abstract-schema-name>

      ...
```

Deployment Descriptor (2)

```
...
<cmp-field>
  <field-name>productID</field-name>
</cmp-field>
<cmp-field><field-name>name</field-name></cmp-field>
<cmp-field>
  <field-name>description</field-name>
</cmp-field>
<cmp-field>
  <field-name>basePrice</field-name>
</cmp-field>
<!--
<primkey-field>productID</primkey-field>
-->
<query>
  ... declações EJB-QL
<query>
</entity>
</enterprise-beans>
...
```



Se PK for um campo em vez de uma classe use este elemento em vez do <prim-key-class>

<query> EJB-QL

```
<query>
  <query-method>
    <method-name>findByName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
  <ejb-ql><![CDATA[ SELECT OBJECT(obj)
  FROM ProductBean obj WHERE obj.name = ?1 ]]></ejb-ql>
</query>
      ...
<query>
  <query-method>
    <method-name>findAllProducts</method-name>
  </query-method>
  <ejb-ql><![CDATA[
   SELECT OBJECT(obj) FROM ProductBean obj
          WHERE obj.productID IS NOT NULL
  ]]></ejb-ql>
</query>
```

Nome de método de Home ou Remote

Parâmetro recebido

Nome de um cmp-field

findByPrimaryKey não precisa ser declarado!

JBoss deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 3.0//EN"
                       "http://localhost/dtd/jboss_3_0.dtd">

<jboss>
  <enterprise-beans>
    <entity>
      <ejb-name>ProductEJB</ejb-name>
      <jndi-name>product/ProductHome</jndi-name>
    </entity>
  </enterprise-beans>
  <resource-managers>
    <resource-manager>
        <res-name>jdbc/ejbPool</res-name>
        <res-jndi-name>java:/Cloudscape</res-jndi-name>
    </resource-manager>
  </resource-managers>
</jboss>
```

JBoss CMP: jbosscmp-jdbc.xml

- *Este arquivo é necessário se a aplicação irá se comunicar com uma fonte de dados existente com um esquema próprio*

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>ProductEJB</ejb-name>
         <table-name>Produto</table-name>
         <cmp-field>
            <field-name>name</field-name>
            <column-name>nome</column-name>
         </cmp-field>
         <cmp-field>
            <field-name>productID</field-name>
            <column-name>codigo</column-name>
         </cmp-field>
         <cmp-field>
            <field-name>basePrice</field-name>
            <column-name>preco</column-name>
         </cmp-field>
            ...
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Tabela que será usada (ou criada) no banco

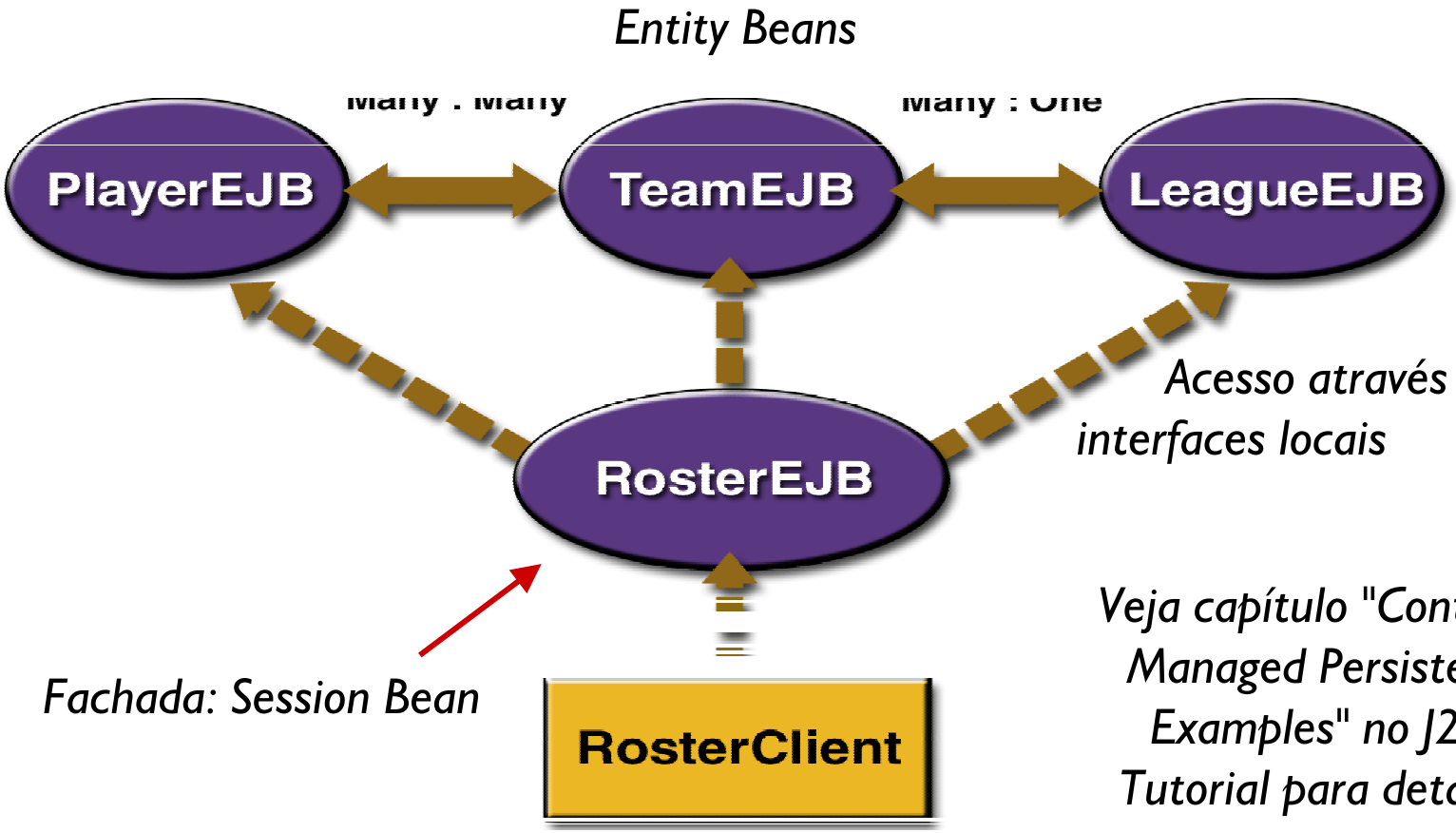
Campos CMP

Campos do banco de dados

Pode-se ainda redefinir o mapeamento de tipos especificado em no arquivo de configuração standardjbosscmp-jdbc.xml

Exemplo com relacionamentos CMR

- Este exemplo utiliza relacionamentos gerenciados pelo container (CMR) que requerem
  - Declaração de métodos de acesso ao relacionamento no Bean
  - Especificação dos relacionamentos no deployment descriptor



Declaração de relacionamentos no Bean

- *PlayerBean*

```
public abstract Collection getTeams();          // many teams
public abstract void setTeams(Collection teams);
```

- *LeagueBean*

```
public abstract Collection getTeams();          // many teams
public abstract void setTeams(Collection teams);
```

- *TeamBean*

```
public abstract Collection getPlayers();        // many players
public abstract void setPlayers(Collection players);
public abstract LocalLeague getLeague();        // one league
public abstract void setLeague(LocalLeague league);
```

Veja DD para Team Bean na próxima página

Relacionamentos no DD para TeamBean

```
<relationships>
  <ejb-relation>                    <description>League-Team</description>
    <ejb-relationship-role>
      <ejb-relationship-role-name>LeagueEJB</ejb-relationship-role-name>
      <multiplicity>One</multiplicity>
      <relationship-role-source>
        <ejb-name>LeagueEJB</ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name>teams</cmr-field-name>
        <cmr-field-type>java.util.Collection</cmr-field-type>
      </cmr-field>
    </ejb-relationship-role>
    <ejb-relationship-role>
      <ejb-relationship-role-name>TeamEJB</ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <cascade-delete />
      <relationship-role-source>
        <ejb-name>TeamEJB</ejb-name>
      </relationship-role-source>
      <cmr-field><cmr-field-name>league</cmr-field-name></cmr-field>
     </ejb-relationship-role>
   </ejb-relation>
            ...
</relationships>
```

campo de relacionamento

Como executar

- *Do Sun J2EE Tutorial: RosterApp*
  - *Diretório cap13/sun/*
- *Configuração*
  - *Configure o arquivo build.properties com seu ambiente*
- *Inicialização*

```
> ant drop.table
> ant clean
```

- *Deployment*

```
> ant jboss.deploy
```
(observe as mensagens de criação das tabelas no JBoss)

- *Execução do cliente*

```
> ant run jboss.client
> ant select.all
```
(faz SELECT na tabela)

Exercícios

- *1. Monte o exemplo RosterApp usando o deploytool da Sun (veja roteiro passo-a-passo em CMP.html)*
- *2. Implemente o exemplo Account do capítulo 11 com CMP*
- *3. Implemente o exemplo Enroller do capítulo 11 com CMP*

Exercício extra

- *4. O JBoss do laboratório está configurado para funcionar com o banco cloudscape. Configure-o para funcionar com o Oracle 8i* instalado na rede*
  - *Copie o arquivo oracle-service.xml fornecido para o diretório deploy do JBoss*
  - *Faça as alterações indicadas no arquivo jbosscmp-jdbc.xml.*
  - *Escolha o prefixo fornecido para seus PK. Implante e rode o exercício 2 (A tabela Contas já está instalada no Oracle).*

Conclusões

- *CMP é uma solução que permite a criação de componentes que são 100% lógica de negócio*
  - *Implementação de persistência resolvida pelo container*
  - *Desenvolvedor se concentra na modelagem e lógica de negócios*
- *Ainda não é possível, sem recorrer a extensões proprietárias, implementar qualquer modelo entidade-relacionamento usando CMP e CMR*
  - *Pode-se usar BMP em parte da aplicação*
  - *Pode-se usar extensões ao EJB-QL nos deployment descriptors (XML) e substituí-los no futuro (usando XSLT)*
  - *EJB 2.1 (2003) resolve principais limitações.*
- *Vimos só a ponta do iceberg. Separação de responsabilidades do CMP abre vários novos horizontes*
  - *Há vários design patterns relativos a CMP/CMR. Veja livro do Floyd Marinescu no CD (EJB Design Pattrerns) e Sun Blue Prints!*

Fontes

[1] *Ed Roman. Mastering EJB 2. 2002*

[2] *Richard Monson-Haefel. Enterprise JavaBeans, 3rd. Edition. O'Reilly, 2002*

[3] *Dale Green, J2EE Tutorial - Container Managed Persistence Examples. Sun Microsystems, 2002*

helder@ibpinet.net

www.argonavis.com.br

Fundamentos básicos de Transações

Helder da Rocha
www.argonavis.com.br

Objetivos

- Apresentar conceitos essenciais sobre transações em aplicações J2EE
  - Este curso não trata de transações em profundidade. Há vários tópicos importantes não mencionados aqui.
  - Para uma abordagem mais profunda, consulte capítulo sobre transações no J2EE Tutorial e no livro Mastering EJB 2 (veja também módulo do curso J530 - EJB, que aprofunda a discussão sobre transações)
- Roteiro
  - O que são transações
  - Transações ACID
  - Container-managed transactions
  - Bean-managed transactions

O que são transações

- Uma transação é uma seqüência de operações que devem ser executadas como se fosse uma só
  - Caso uma das operações falhe, todo o processo é abortado e revertido ao estado anterior ao início da transação
- Uso típico consiste do emprego de dois métodos
  - commit(): usado após a declaração das instruções a serem executadas
  - rollback(): chamado quando há uma falha. O método rollback() deve desfazer todas as operações iniciadas pelas instruções que foram executadas
- Há duas formas de controlar transações em J2EE
  - CMT - Container Managed Transactions: container administra todo o processo por método - aspect-oriented approach.
  - BMT - Bean Managed Transactions: bean chama as APIs diretamente e controla início e fim das transações

ACID

- *Características essenciais de uma transação:*
  ***A**tomic, **C**onsistent, **I**solated, **D**urable*
- *Atomic*
  - *Todas as tarefas de uma unidade transacional devem funcionar sem erros ou todo o processo é revertido*
- *Consistent*
  - *O estado do sistema após uma transação deve manter-se consistente (transações devem englobar processos de negócio completos)*
- *Isolated*
  - *Transação deve poder executar sem interferência de outros processos*
- *Durable*
  - *Dados alterados durante a transações devem ser guardados em meio persistente até que a transação complete com sucesso*

CMT: JTS declarativo

- *Container-Managed Transactions*
  - *Controle de transações totalmente gerenciado pelo container*
  - *Não permite o uso de métodos commit() e rollback() dentro do código*
  - *Única forma de controlar transações em Entity Beans*
- *Deployment descriptor*
  - *Declaração do tipo de transação do bean: <ejb-jar>/ <enterprise-beans> / <session>*:*

```
<transaction-type>Container</transaction-type>
```

  - *Descrição do atributo para cada método*

```
<assembly-descriptor>
    <container-transaction>
        <method> ... </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
    (...)
</assembly-descriptor>
```

* *ou message-driven*

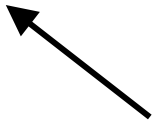
Atributos (políticas transacionais)

- *NotSupported*
  - *Chamada de um método com este atributo suspende a transação até que o método termine*
- *Supports*
  - *Método será incluído no escopo de uma transação se for chamado dentro de uma transação*
- *Required*
  - *Método tem que ser chamado no escopo de uma transação*
- *RequiresNew*
  - *Nova transação sempre é iniciada quando este método for chamado*
- *Mandatory*
  - *Método sempre deve fazer parte do escopo de uma transação*
- *Never*
  - *Método não deve ser chamado no escopo de uma transação*

Os livros-texto discutem as implicações desses atributos - leia!

JTS declarativo - exemplo

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>BankEJB</ejb-name>
            <home>j2eetut.bank.BankHome</home>
            <remote>j2eetut.bank.Bank</remote>
            <ejb-class>j2eetut.bank.BankBean</ejb-class>
            <session-type>Stateful</session-type>
            <transaction-type>Container</transaction-type>
            (...)
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>BankEJB</ejb-name>
                <method-name>getSavingBalance</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction> (...)
    </assembly-descriptor>
</ejb-jar>
```



bank-ejb-jar.xml

BMT - controle explícito de transações

- *Bean-Managed Transaction: JTA pode ser usada por session beans e message-driven beans (BMT não pode ser usada por Entity Beans). Deployment descriptor:*

```
<transaction-type>Bean</transaction-type>
```

- *Transações JDBC*
  - *commit/rollback em java.sql.Connection*
- *JTS: Java Transaction Service - implementação Java de OTS (Object Transaction Service)*
  - *Para programar diretamente em JTS é preciso usar sintaxe CORBA*
- *JTA: Java Transaction API*
  - *Mais simples que programar diretamente em JTS*
  - *Dois componentes: interface cliente de alto nível e interface XA (para transações distribuídas) de baixo nível*
  - *Principal interface do JTA:* `javax.transaction.UserTransaction`

Transações explícitas JTA - exemplo

```
(...)
   public void withdrawCash(double amount) {
      UserTransaction ut = sessionCtx.getUserTransaction();
      try {
         ut.begin();
          updateChecking(amount);
          machineBalance -= amount;
          insertMachine(machineBalance);
         ut.commit();
      } catch (Exception ex) {
          try {
             ut.rollback();
          } catch (SystemException syex) {
              throw new EJBException
                 ("Rollback failed: " + syex.getMessage());
          }
          throw new EJBException
             ("Transaction failed: " + ex.getMessage());
     }
   }
```

TellerBean.java

Transações JDBC - exemplo

```
(...)
public void ship (String productId,
                  String orderId,
                  int quantity) {
  try {
     con.setAutoCommit(false); // con é java.sql.Connection
       updateOrderItem(productId, orderId);
       updateInventory(productId, quantity);
     con.commit();
  } catch (Exception e) {
    try {
      con.rollback();
      throw new EJBException("Transaction failed: "
                                          + e.getMessage());
    } catch (SQLException sqx) {
      throw new EJBException("Rollback failed: "
                                          + sqx.getMessage());
    }
  }
}
```

WarehouseBean.java

Transações: riscos

- O uso de transações oferece alguns riscos
  - Impacto significativo na performance
  - Deadlock
- Em EJB o risco de deadlock é minimizado pela proibição de uso de threads, e uso correto dos políticas transacionais (Required, RequiresNew,etc.)
  - Mantenha o sistema simples: use transações apenas nos métodos onde isto é realmente necessário
  - Use níveis de isolamento fracos quando possível (isto depende do seu banco de dados)
- A correta escolha entre BMT/CMT, políticas e níveis de isolamento de cada método permitem amplo controle sobre performance x segurança dos dados.

Exemplos

- *Os exemplos deste capítulo são do J2EE Tutorial*
- *Para executá-los, configure build.properties e rode o ant com os targets:*
  - *jboss.deploy* — *implanta os beans no JBoss*
  - *run.container.jts.client* — *roda aplicação CMT (Bank)*
  - *run.bean.jdbc.client* — *roda aplicação BMT - JDBC (Warehouse)*
  - *run.bean.jta.client* — *roda aplicação BMT - JTA (Teller)*

Fontes

[1] *Richard Monson-Haefel. Enterprise JavaBeans, 3rd. Edition. O'Reilly and Associates, 2001*

[2] *Dale Green. J2EE Tutorial - Transactions. Sun J2EE Tutorial, 2002*

helder@argonavis.com.br

argonavis.com.br

Fundamentos básicos de Segurança

Helder da Rocha
www.argonavis.com.br

Objetivos

- *Este capítulo apresenta*
  - *Noções básicas de segurança em J2EE*
  - *Exemplos de aplicações*
  - *Usos típicos de JAAS*
  - *Configuração básica de aplicações J2EE com recursos de segurança*
- *A abordagem deste assunto neste curso é superficial. Para maiores detalhes consulte*
  - *Livros-texto (MEJB contém versão original do exemplo mostrado usando outro modelo de autenticação; J2EE Tutorial mostra exemplo usando container do J2EE-RI)*
  - *Código-exemplo (comentado)*

Controle de acesso

- *J2EE define uma especificação para controle de acesso e autenticação de clientes*
  - *Identidade é representada por objeto java.security.Principal*
  - *Não há uma recomendação formal para outros aspectos importantes de segurança, tais como criptografia e comunicação segura*
- *O controle de acesso em aplicações J2EE é feito de forma declarativa nos arquivos web.xml e ejb-jar.xml*
  - *Declara-se papéis (roles) lógicos que podem ser mapeados a usuários e grupos em tempo de deployment: <security-role>*
  - *Papéis são associados a métodos para determinar quem tem autorização para executar: <method-permission>*
  - *Métodos podem ser executados em outros beans por um Principal diferente daquele que executa o bean atual: <security-identity>*

Exemplo

```
<assembly-descriptor>
  <security-role>
    <role-name>BankCustomer</role-name>
  </security-role>
  <security-role>
    <role-name>BankAdmin</role-name>
  </security-role>
  <method-permission>
    <unchecked />
    <method>
      <ejb-name>CustomerEJB</ejb-name>
      <method-name>getPrimaryKey</method-name>
    </method>
  </method-permission>
  <method-permission>
    <role-name>BankAdmin</role-name>
    <method>
      <ejb-name>CustomerEJB</ejb-name>
      <method-name>*</method-name>
    </method>
  <method-permission>
</assembly-descriptor>
```

ejb-jar.xml

Outros métodos de CustomerEJB só podem ser chamados por usuários que assumem o papel de BankAdmin

JAAS

- Java Authentication and Authorization Service
  - Conjunto de pacotes para autenticação de usuários e autorização
  - Maior parte implementada pelo servidor
- Principais classes
  - javax.security.auth.Subject
  - javax.security.Principal
  - javax.security.auth.callback.Callback
  - javax.security.auth.callback.CallbackHandler
  - javax.security.auth.login.Configuration
  - javax.security.auth.login.LoginContext
  - javax.security.auth.spi.LoginModule

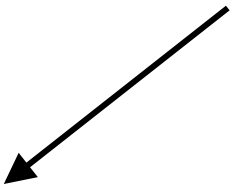
Implementação do JAAS no JBoss (JBossSX)

- A implementação do JAAS no JBoss consiste de
  - JAASSecurityManager
  - Módulos de configuração do servidor (login-config.xml) e do cliente (auth.conf)
  - Implementações de javax.security.auth.spi.LoginModule
- ClientLoginModule
  - Implementação de LoginModule no cliente
- UsersRolesLoginModule
  - Uma das implementações de LoginModule no servidor
  - Par de arquivos de texto para definir usuários e grupos
- jboss.xml e jboss-web.xml
  - Declaram domínio JAAS a ser usado através do elemento <security-domain> de jboss.xml e jboss-web.xml

Exemplo: ejb-jar.xml

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>SecureHelloEJB</ejb-name>
            <home>examples.HelloHome</home>
            <remote>examples.Hello</remote>
            <ejb-class>examples.HelloBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <security-role>
            <role-name>TestRole</role-name>
        </security-role>
        <method-permission>
            <role-name>TestRole</role-name>
            <method>
                <ejb-name>SecureHelloEJB</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
    </assembly-descriptor>
</ejb-jar>
```

Todos os métodos acessíveis a TestRole



Veja código exemplo em cap15/src

Serviço JBossSX com UsersRolesLoginModule

- *Módulo simples que utiliza par de arquivos de texto*
  - *É a configuração default em login-config.xml*

```
<policy>  (...)
 <application-policy name="other">
  <authentication>
   <login-module
     code="org.jboss.security.auth.spi.UsersRolesLoginModule"
     flag="required" />
  </authentication>
 </application-policy>
</policy>
```

Trecho de server/default/conf/login-config.xml

users.properties (nome=senha)

```
mickey=mouse
niquel=nausea
```

Use preferencialmente senha criptografada!

roles.properties (nome.Grupo=Role,...,Role)

```
mickey.Roles=TestRole, AdminRole
niquel.Roles=NoRole, MouseRole
```

Coloque no CLASSPATH do servidor (pode ser no EJB-JAR)

Domínio de segurança (servidor)

- *Deve ser declarado em jboss.xml (ou jboss-web.xml)*

```
<jboss>
   <security-domain>
     java:/jaas/TestServerDomain</security-domain>
   <session>
      <ejb-name>SecureHelloEJB</ejb-name>
      <jndi-name>login/HelloHome</jndi-name>
   </session>
</jboss>
```

jboss.xml

- *Deve coincidir com nome de um <application-policy> em login-config.xml*

```
<policy> (...) <application-policy name="TestServerDomain">
   <authentication>
       <login-module code="MyLoginModule" .../>
```

- *Se não houver domínio com mesmo nome em login-config.xml, será usado o domínio default "other", previamente configurado para usar UsersRolesLoginModule (veja slide anterior)*

Domínio usando bando de dados

- Pode-se guardar nome, senha e grupo em banco de dados. Para isto, é preciso configurá-lo no JBoss (login-config.xml) com DatabaseServerLoginModule
  - É preciso também ter uma ou mais tabelas criadas com os dados de login (usuario, senha e grupo)
- O módulo requer três opções que são: nome do Datasource, query que retorne a senha, dado o userid e query que retorne um grupo, como "Roles" dado um userid

```
<application-policy name="dominio-JDBC-servidor">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule">
      <module-option name="dsJndiName">java:/DefaultDS</module-option>
      <module-option name="principalsQuery">
           select senha from usuarios where id=?</module-option>
      <module-option name="rolesQuery">
           select grupo, 'Roles' from grupos where id=?</module-option>
      <module-option name="unauthenticatedIdentity">nobody</module-option>
       </login-module>
    </authentication>
 </application-policy>
```

Domínio do cliente JBossSX

- *Coloque no Classpath do cliente*
  - *arquivo de configuração (indica no mínimo a implementação de login module usada)*

arquivo de configuração (default)

```
dominio-cliente {
    // JBoss LoginModule implementation
    org.jboss.security.ClientLoginModule  required;
};
```

  - *Informe ao cliente a localização do arquivo através da propriedade java.security.auth.login.config:*

```
> java Prog -Djava.security.auth.login.config=auth.txt
```

  - *JARs do JBossSX: jbosssx-client.jar (além dos outros JARs necessários para o cliente JBoss)*

Domínio de clientes dentro de containers

- *Para clientes que executam dentro do servidor (ex: servlets, JSP), chame o mesmo login module através de domínio definido na configuração do JBoss*
- *Ex: domínio declarado em login-config.xml:*

```
<application-policy name="dominio-cliente">
  <authentication>
    <login-module
        code="org.jboss.security.ClientLoginModule"
        flag="required">
    </login-module>
   </authentication>
</application-policy>
```

Cliente

```
public class HelloClient {
  public static void main(String[] args) throws Exception {
    LoginContext loginCtx = null;
    try {
      // Cria CallBackHandler
      CallbackHandler handler = new HelloCallbackHandler();
      // Carrega configuração
      loginCtx = new LoginContext("dominio-cliente", handler);
      // Faz o login
      loginCtx.login();
    } catch (LoginException e) {
      System.out.println("Login failed"); System.exit(1);
    }
    // Executa ação privilegiada propagando contexto de segurança
    Context ctx = new InitialContext(System.getProperties());
    Object obj = ctx.lookup("SecureHelloEJB");
    HelloHome home = (HelloHome)
              PortableRemoteObject.narrow(obj, HelloHome.class);
    Hello hello = home.create();
    System.out.println(hello.hello());
  }
}
```

Encapsula dados de autenticação

Dominio do cliente JBossSX (veja slides anteriores)

Faz o login aqui

Chama métodos privilegiados

CallbackHandler

```
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;

public class HelloCallbackHandler implements CallbackHandler {
    private String username;
    private String password;

    public HelloCallbackHandler() {
        LoginDialog dialog = new LoginDialog("User Authentication Required");
        this.username = dialog.getData()[0];
        this.password = dialog.getData()[1];
    }
    public void handle(Callback[] callbacks)
                        throws java.io.IOException, UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof NameCallback) {
                NameCallback nc = (NameCallback) callbacks[i];
                nc.setName(username);
            } else if (callbacks[i] instanceof PasswordCallback) {
                PasswordCallback pc = (PasswordCallback) callbacks[i];
                pc.setPassword( this.preparePassword(password) );
            } else throw new UnsupportedCallbackException(callbacks[i], "Error");
        }
    }
    private char[] preparePassword(String word) {
        return word.toCharArray(); // ou rotina de criptografia
    }
}
```

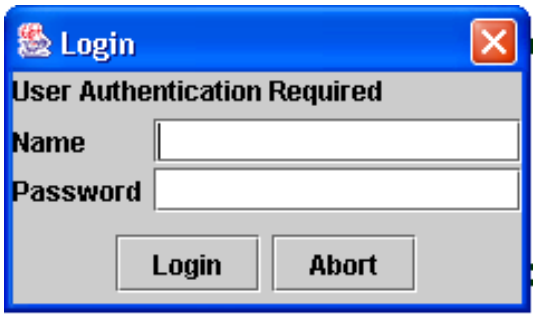
GUI simples para entrada de texto

Como executar

- Diretório cap15/mejb
- Configuração
  - Configure build.properties
  - Edite usuários e grupos em lib/users.properties e lib/roles.properties
- Use targets do Ant

  `> ant jboss.deploy` (para instalar)

  `> ant run.jboss.client` (para rodar)
- Digite nome e senha compatíveis com arquivos de configuração
  - Tente logar como usuário válido não autorizado (niquel)



Exercício

- *1. a) Crie uma aplicação contendo um session bean com dois métodos*
  - *listarNomes()*
  - *adicionarNome(String nome)*
- *1. b) Crie dois papéis (roles): User e Admin e defina as permissões dos métodos*
  - *Admin: pode listarNomes() e adicionarNome()*
  - *User: pode listarNomes()*
- *1. c) Implemente um cliente que faça o login e chame um dos dois métodos*
- *1. d) Configure usuários do sistema, associe-os aos papéis existentes e rode a aplicação*

Use como esqueleto os exemplos fornecidos!

Fontes

[1] *JBoss Group. JBoss User's Manual. Chapter 8: The JBoss Security Extension Framework.*

[2] *Erik Jendrock. Security. Sun J2EE Tutorial.*

[3] *Richard Monson-Haefel, Enterprise JavaBeans, 3rd. Edition, O'Reilly and Associates, 2001*

[4] *Ed Roman. Mastering EJB 2.*

helder@argonavis.com.br

argonavis.com.br

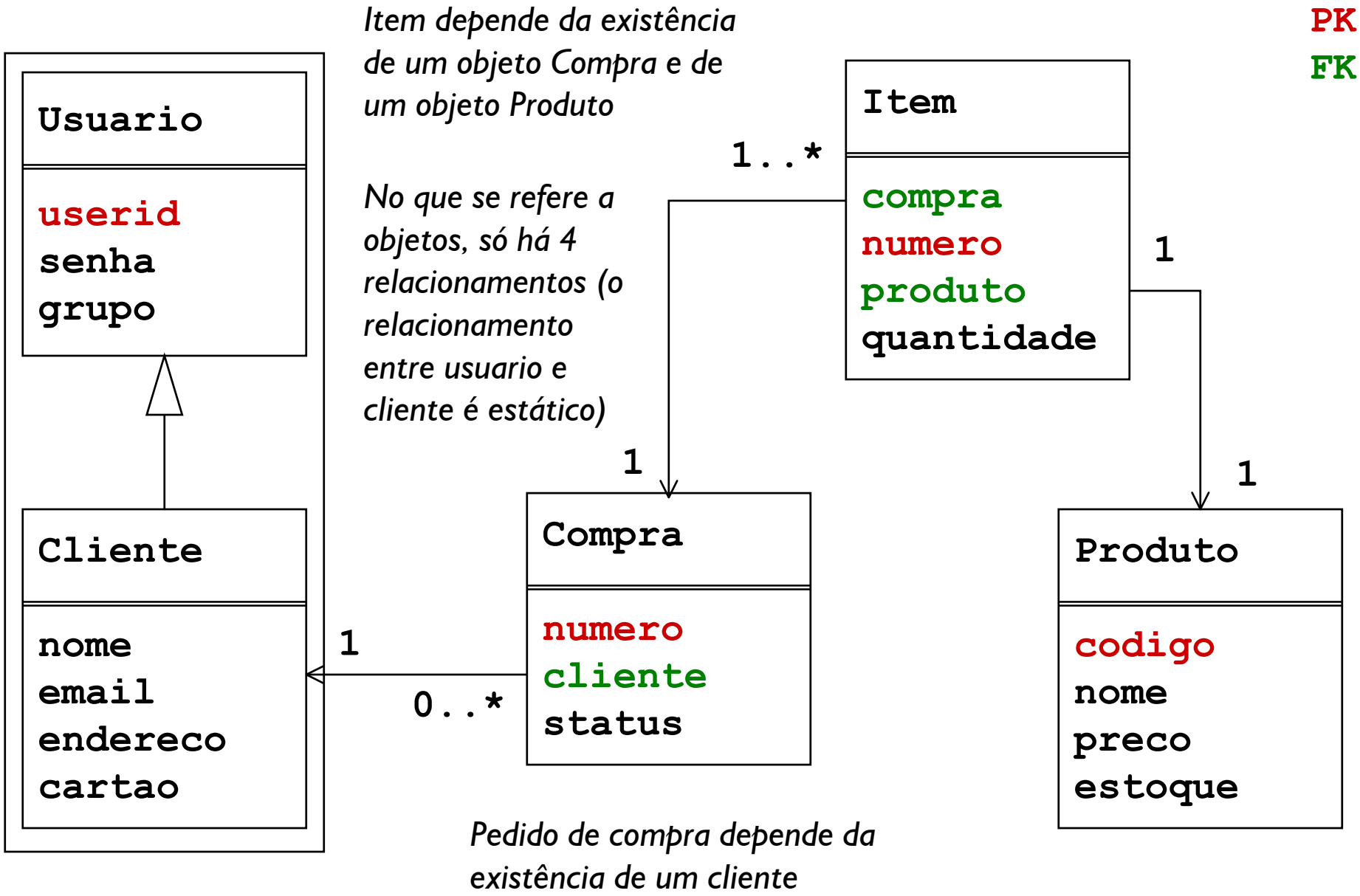
Uma aplicação J2EE completa

Helder da Rocha
www.argonavis.com.br

Objetivos

- *O objetivo deste módulo é construir e implantar uma aplicação J2EE completa*
- *Inicialmente, será mostrada como exemplo uma aplicação consistindo de vários beans, JSPs e servlets usando serviços de transações e segurança*
  - *A aplicação utiliza alguns padrões de projeto J2EE*
  - *Todos os passos de compilação e montagem estão embutidos no build.xml*
- *Diferentemente dos outros exemplos do curso, estas aplicações utilizarão o banco HSQLDB, que é o banco nativo do JBoss*

Aplicação exemplo: relacionamentos



Aplicação exemplo: tabelas no banco (I)

```
create table produtos (
        id integer primary key,
        nome varchar(64),
        preco numeric(10,2),
        qte   integer
);

create table usuarios (
        id varchar(8) primary key,
        senha varchar(16),
        grupo varchar(16)
);

create table clientes (
        id varchar(8) primary key,
        nome varchar(64),
        email varchar(64),
        endereco varchar(256),
        cartao varchar(16),
        constraint fk_usuario  foreign key (id)
                               references usuarios(id)
);
```

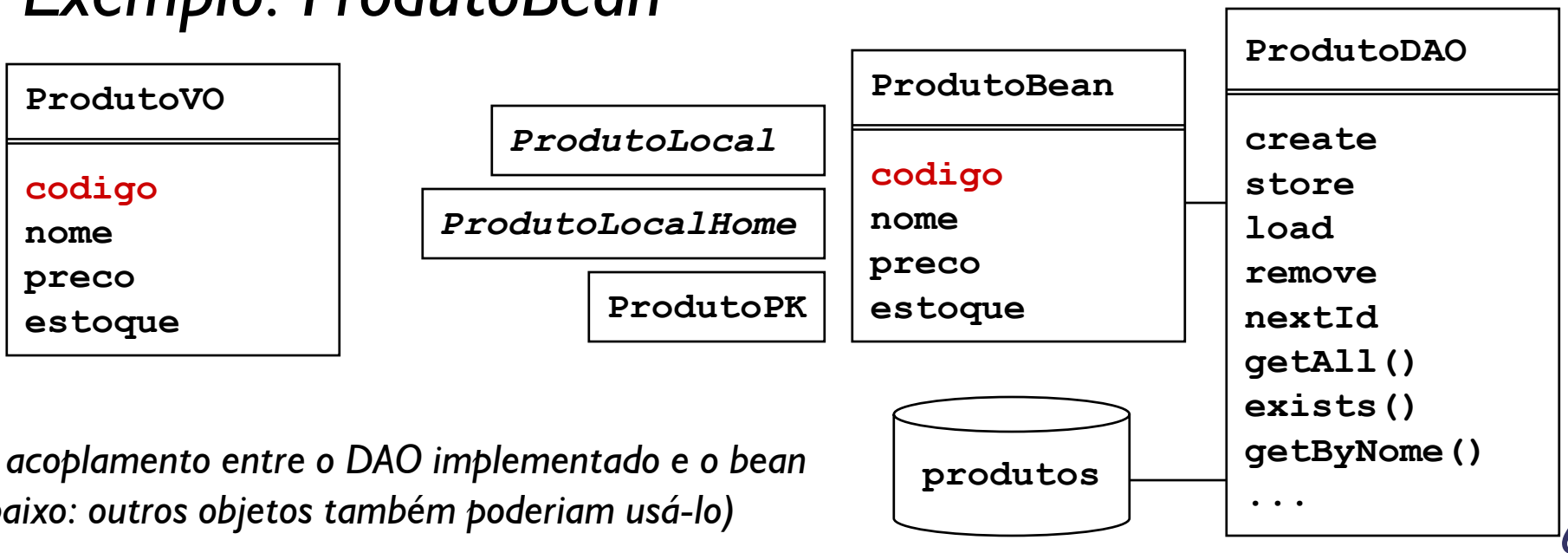
Aplicação exemplo: tabelas no banco (2)

```
create table compras (
        id integer primary key,
        cliente varchar(8),
        status varchar(16),
        constraint fk_cliente  foreign key (cliente)
                               references clientes(id)
);

create table itens (
        id integer primary key,
        compra integer,
        produto integer,
        qte integer,
        constraint fk_compra   foreign key (compra)
                               references compras(id),
        constraint fk_produto  foreign key (produto)
                               references produtos(id)
);
```

Poderia ser melhor implementado com PK composta de compra + id

Aplicação exemplo: entity beans

- Cada entity bean (BMP) possui
  - Um par de interfaces locais e uma classe PK
  - Um Value Object: objeto serializável que pode armazenar seu estado (também chamado de Data ou Transfer object)
  - Um DAO: objeto que encapsula a lógica de persistência e é chamado a partir dos métodos callback do bean
- Exemplo: ProdutoBean



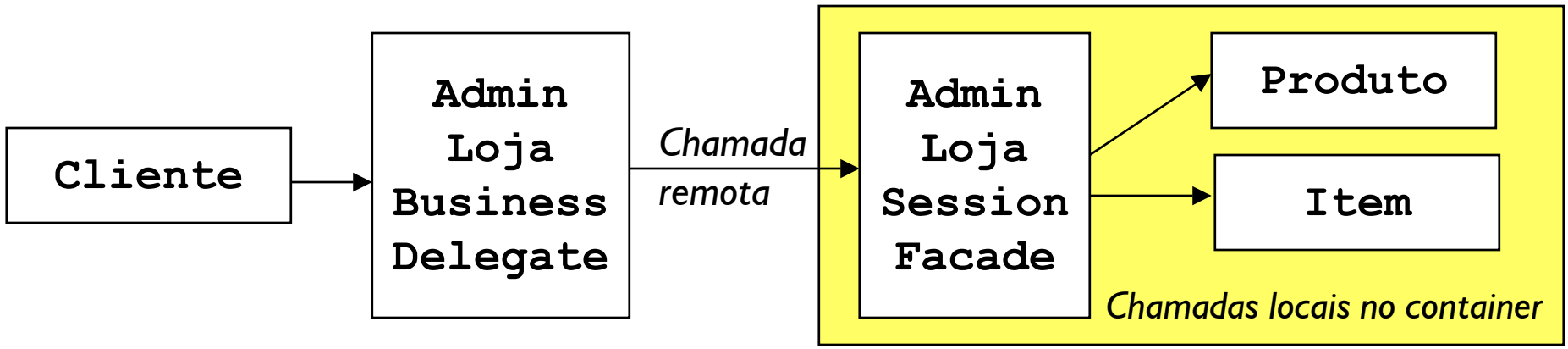
(O acoplamento entre o DAO implementado e o bean é baixo: outros objetos também poderiam usá-lo)

Aplicação exemplo: session beans

- Session beans implementam todas as operações da aplicação. Três são fachadas:
  - AdminLojaSessionFacade: operações de administração da loja: criar produtos, aumentar estoque, etc. (Stateless)
  - LoginSessionFacade: operações de autenticação, criação e alteração de clientes cadastrados. (Stateless)
  - LojaSessionFacade: representa a sessão de um cliente fazendo compras na loja (Stateful)
- Parte de LojaSessionFacade é um carrinho de compras, que é implementando em outro bean
  - CarrinhoSession: contém operações do carrinho de compras (Stateful)

Aplicação exemplo: fachadas

- *Os session beans isolam o cliente da lógica de negócios (entity beans) implementando todas as operações*
  - *São acessados remotamente pelos clientes, portanto, têm um par de interfaces remotas*
- *Cada bean que implementa operações de fachada possui um Business Delegate, que isola o cliente da tecnologia usada na implementação da aplicação*
  - *Cliente não recebe exceções EJB ou sequer precisa saber da API javax.ejb. Business Delegate captura essas exceções e as inclui em exceções da aplicação (LojaException)*



Aplicação exemplo: message-driven beans

- A aplicação possui três MDBs
  - ControleEstoqueMDB: recebe pedidos para aumentar o estoque. Os pedidos chegam em uma fila usada pelo AdminLojaSessionFacade no metodo solicitarCompra()
  - ProcessarCompraMDB: recebe dados de uma compra e os utiliza para processá-la (verificar cartão de crédito, etc.) depois, altera o status da compra (para concluido ou suspenso) e passa as informações para o MailerMDB
  - MailerMDB: usa a API JavaMail para enviar um e-mail ao cliente informando se seu pedido foi processado ou não.

Aplicação exemplo: Service Locator

- A aplicação isola os beans e clientes (delegates) de precisarem usar a API JNDI oferecendo métodos utilitários que devolvem beans, filas, conexões de bancos de dados e variáveis de ambiente
  - Foi implementado um único ServiceLocator genérico (não um por bean - que é uma outra estratégia)
  - No ServiceLocator genérico, os métodos requerem que se passe o caminho JNDI dos objetos a serem localizados
- Métodos
  - java.sql.Connection findJDBCConnection(nome)
  - javax.jms.Connection findDefaultJMSConnection()
  - javax.jms.Queue findJMSQueue(nome, nomeAlternativo)
  - javax.ejb.EJBHome findRemoteEJB(nome, classe)
  - javax.ejb.EJBLocalHome findLocalEJB(nome)
  - java.lang.Object findEnvEntry(nome)

Aplicação exemplo: camada Web

- A camada Web da aplicação exemplo usa três servlets controladores e comandos
  - FrontController & Command pattern
- Também usa os Value Objects da aplicação EJB como View Helpers (para preencher os JSPs) e uma pequena tag library
- Os servlets são clientes e interagem com os Business Delegates
- Não foi usada autenticação Web
  - Usamos autenticação EJB através do LoginBusinessDelegate

Como implantar e executar

- 1. Verifique a configuração do seu ambiente inspecionando o arquivo build.properties
- 2. Crie primeiro as tabelas usando
  > ant create-table
- 3. Depois monte a aplicação com
  > ant deploy
- 4. Para executar os clientes EJB, use
  > ant run.all.test.clients
- 5. Para acessar a aplicação via Web, use o browser
  - http://localhost:8080/loja/
  - Cadastre primeiro um usuário do tipo admin para criar alguns produtos.
  - Inicie o James ou outro servidor de e-mail local.
- Gere documentação (no diretório docs) usando
  > ant generate-docs

Exercício

- Neste exercício você vai implementar uma parte da aplicação demonstrada em sala de aula.
  - Use e configure seus build.xml para que você possa compilar e verificar os arquivos gerados continuamente, perdendo menos tempo com depurações complicadas.
  - Use os arquivos e estrutura de pacotes fornecida!
- O exercício consiste de duas partes:
  - 1. Criação de uma aplicação EJB usando um Entity Bean (BMP), um Session Bean e um Message-driven bean
  - 2. (se der tempo*) Adaptação da aplicação acima para a Web usando JSP, servlets e custom tags. Neste exercício, parte do código já está pronto.

* Se o tempo for curto, use a implementação pronta e integre-a à aplicação EJB

Parte I - Exercício I

- *a) Crie um Entity Bean com interfaces locais que esteja sincronizado com os dados da tabela de produtos do banco.*
  - *Não coloque código JDBC no bean. Use o DAO fornecido com métodos create/load/store/remove e chame os métodos do DAO de dentro do bean.*
  - *Use também ProdutoVO (value object) para transferir dados. O DAO preenche esse bean e o EJB pode recebê-lo como parâmetro ou retorná-lo em seus métodos.*
- *(b) Configure o seu ejb-jar e use o verifier para validar o seu EJB-JAR (use a tarefa do Ant).*
- *(c) Se tudo estiver OK, faça deploy e observe eventuais erros.*
- *(d) Escreva um cliente simples para testar o seu EJB*
  - *Para isto, registre (no ejb-jar.xml) e use também um par de interfaces remotas (somente com esta finalidade)*

Parte I - Exercício 2

- *a) Implemente as operações abaixo (definidas na classe AdminLojaBusinessDelegate) em um session bean. Faça com que o BD localize e chame os métodos do session bean*
  - *criarProduto(String nome, BigDecimal preco): String*
  - *removerProduto(String codigo)*
  - *ProdutoVO[] listarProdutos()*
  - *solicitarCompra(String codigo, int quantidade)*
  - *esvaziarEstoque(String codigo)*
- *b) Acesse o EJB de produto através de uma interface local.*
  - *Implemente os métodos acima usando Produto*
  - *Não se esqueça de configurar no ejb-jar.xml as referências para o produto a partir do outro bean.*
  - *Mapeie referências locais a nomes JNDI reais no arquivo jboss.xml.*
  - *Use, se desejar, o service locator fornecido.*
- *c) Use o cliente fornecido para testar a aplicação*

Parte I - Exercício 3

- *a) Crie um MDB (ControleEstoqueMDB) que receba pedidos de aumento de estoque assíncronamente através da escuta de uma fila JMS (use queue/testQueue).*
  - *A mensagem enviada deve conter o código do produto e a quantidade desejada. O MDB deve, então, localizar o bean (Produto) e fazer a alteração.*
  - *Configure o bean no ejb-jar.xml, verifique e faça o deployment.*
  - *Não se esqueça de registrar os nomes JNDI globais de todos os beans, filas e bancos de dados no jboss.xml.*
- *b) Altere o método solicitarCompra para que ele em vez de alterar o produto diretamente, envie para a fila onde está cadastrado o MDB para que ele leia e processe o pedido.*
- *c) Rode o cliente e teste a aplicação.*

Parte II

- *Objetivo: adaptar a aplicação anterior para que funcione também via Web.*
- *a) Use os três custom tags fornecidos (que já são automaticamente compilados e empacotados pelo Ant) no classpath do WAR, os JSP que estão praticamente prontos (falta só as chamadas nos formulários e links), o servlet e os comandos.*
- *b) preencha os comandos com chamadas ao BusinessDelegate para que a aplicação funcione na Web,*
- *c) defina o fluxo de controle da aplicação no mapping.xml,*
- *d) empacote o WAR e configure o web.xml com referências para os EJBs, em um EAR e faça o deployment.*

Não havendo tempo, pegue a parte Web pronta da solução e integre-a com sua aplicação

Exercícios extras (para turmas avançadas)

- *1. Converta a aplicação da parte I em CMP*
- *2. (na aplicação exemplo) Inclua todo o processo de compra em uma transação distribuída para que, se o usuário cancelar o processo em qualquer etapa, quaisquer tabelas criadas sejam destruídas*
  - *Jamais deve restar na tabela compras registros com status "Iniciado"*
- *3. (na aplicação exercício) Implemente autenticação e autorização na aplicação e defina privilégios de administrador aos métodos que alteram os dados da aplicação.*
- *4. (na aplicação exemplo) Inclua o processo de compra, inclusive a parte assíncrona (MDB)) em uma transação distribuída e garanta que pedidos rejeitados (status: "Suspenso") sejam desfeitos*
- *5. Converta a aplicação exemplo em CMP com CMR*

helder@argonavis.com.br

www.argonavis.com.br

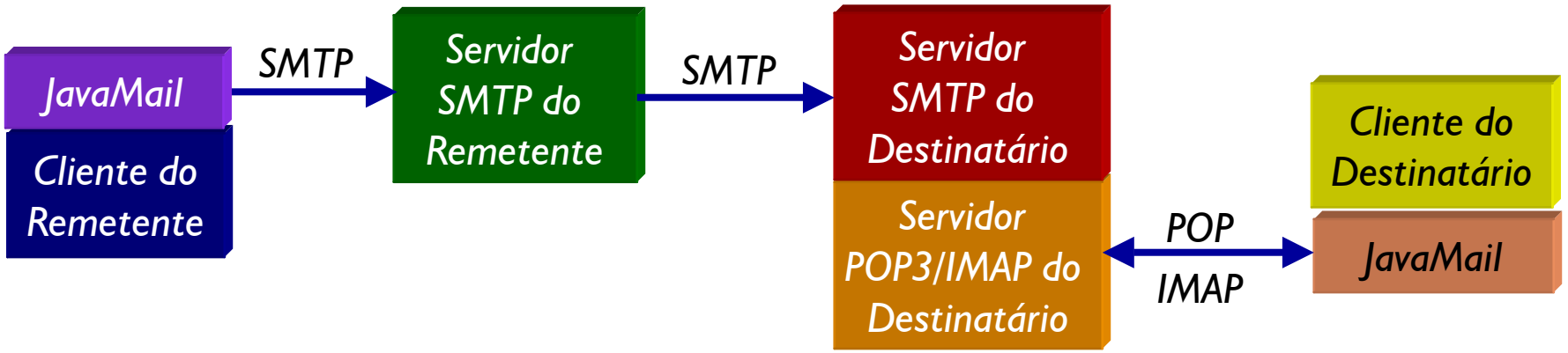
Helder da Rocha
www.argonavis.com.br

O que é JavaMail?

- API genérica para construir aplicações que manipulam correio eletrônico (não necessariamente Internet e-mail) e messaging como clientes
  - Serve para estender aplicações existentes para lidar com mensagens, cabeçalhos, attachments, etc.
  - Permite um nível de abstração mais alta (em comparação com sockets)
  - Permite independência de implementação (em comparação alternativas como "com.sun.net.smtp.*"
- Distribuição oferece suporte a MIME, SMTP, IMAP, POP, etc. mas pode ser estendida

Arquitetura de sistemas de e-mail

- SMTP - Simple Mail Transfer Protocol
  - Principal meio de transporte para e-mail na Internet
- POP - Post Office Protocol
  - Lida com repositórios (Stores) de e-mails recebidos
  - Cliente POP pode se conectar a servidor para baixar mensagens recebidas
- IMAP
  - Pode ser usado no lugar do POP



JavaMail 1.2 API

- Oferece uma camada abstrata sobre diversas implementações de correio eletrônico
  - Permite criar clientes genéricos
- Principais classes da API
  - Sessões: javax.mail.Session
  - Serviço de Armazenamento: javax.mail.Store
  - Serviço de transporte: javax.mail.Transport
  - Mensagens: javax.mail.Message, Part, MultiPart
- Maior parte da API são interfaces e classes abstratas
  - Há implementações para SMTP, IMAP, NNTP*, POP3
- JARs
  - activation.jar (JavaBeans Activation Framework)
  - mail.jar

* parcial

A Sessão

- *Dois papéis*
  - *guardar propriedades acessíveis via construtores*
  - *oferece métodos de fábrica para objetos Store e Transport*
- *Dois tipos de objetos Session*

```
Session session = Session.getDefaultInstance(props)
Session session Session.getInstance(props)
```

Propriedades como servidor de email, usuário, etc.

- *Autenticadores*
  - *Passados como parâmetro nos métodos getInstance()*

```
public class MyAuthenticator extends Authenticator {
  protected PasswordAuthentication getPasswordAuthentication() {
     return new PasswordAuthentication("scott", "tiger");
  }
}
(...)
Properties props = System.getProperties();
Session session =
     Session.getDefaultInstance(props, new MyAuthenticator());
```

Mensagens

- São encapsuladas na classe abstrata Message, que possui
  - Conjunto de cabeçalhos RFC822 (Headers)
  - Conteúdo (javax.activation.DataHandler)
- Há métodos get/set para conteúdo e cabeçalhos
- Distribuição JavaMail traz uma implementação da classe abstrata Message: MimeMessage

```
Message msg = new MimeMessage(session);
```

- Destinos e origens são definidos em objetos do tipo Address e adicionados nos cabeçalhos To e From

```
msg.setFrom(new InternetAddress("fulano@destino.org"));
msg.setRecipients(Message.RecipientType.TO,
                  InternetAddress.parse(recv, false));
msg.setSubject("JavaMail Test Message");
msg.setHeader("X-Mailer", "My Simple JavaMail Client");
msg.setText("This is the sample message text");
```

Envio de mensagens

- *Depois de criada a mensagem, basta passá-la para um meio de transporte para enviá-la*
- *Forma mais simples de fazer é passar para o método (estático) send(), de javax.mail.Transport*

```
Transport.send(msg);
```

- *Pode-se também controlar o tipo de transporte usado (de acordo com o protocolo)*

```
Transport smtp = session.getTransport("smtp");
smtp.send(msg);

Transport nntp = session.getTransport("nntp");
nntp.send(msg);
```

Recuperação de Mensagens

- *JavaMail 1.2 possui implementações para POP3 e IMAP*
- *A classe Store permite a conexão a um repositório de mensagens (message store)*

```
Store popStore = session.getStore("pop3");
popStore.connect("localhost", null, null);
```

- *Obtendo-se um objeto Store, é preciso buscar a pasta default e, a partir dela, a pasta desejada (por exemplo: INBOX).*
- *Depois é preciso abrir a pasta para leitura ou leitura/gravação*

```
Folder defaultPopFolder = popStore.getDefaultFolder();
inboxFolder = defaultPopFolder.getFolder("INBOX");
inboxFolder.open(Folder.READ_ONLY);
```

- *A partir daí, pode-se ler as mensagens*

```
Message[] msgs = inboxFolder.getMessages();
```

Filtros

- *A classe abstrata SearchTerm permite fazer buscas em mensagens dos folders com base em critérios dependentes da implementação de uma subclasse*
  - *SubjectTerm, faz buscas baseadas no assunto*
  - *FromStringTerm busca no cabeçalho From*
  - *AndTerm, OrTerm, NotTerm aceitam outros SearchTerm*

```
SearchTerm st =
  new AndTerm(new FromStringTerm("elfodon@gurus.org"),
              new SubjectTerm("[java-list]"));
Message[] msgs = folder.search(st);
```

- *Utilizando SearchTerms, pode-se filtrar mensagens indesejadas e apagá-las dos Folders*

```
SearchTerm st = new SubjectTerm("Ganhe dinheiro!");
Message[] msgs = folder.search(st);
msgs[0].setFlag(Flags.Flag.DELETED, true);
inboxFolder.close(true);
```

Sinaliza para remoção de mensagem

Recebimento de mensagens com anexos

- Mensagens podem ter uma ou várias partes. Pode-se compor uma mensagem com várias outras mensagens
- Mensagens multipartes (Multipart) consistem de uma série de objetos BodyPart.
  - Cada BodyPart tem seu próprio tipo de dados MIME e conteúdo.
  - A mensagem tem timpo MIME que começa com "multipart/"
- Ao obter uma mensagem do folder, verifica-se se a mensagem é do tipo "multipart/*". Se for
  - Obtenha o conteúdo (objeto Multipart)
  - Use mp.getBodyPart(idx) para obter cada MimeBodyPart

```
if (msg.isMimeType("multipart/*")) {
   Multipart mp = (Multipart) msg.getContent();
   for (int i = 0; i < mp.getCount(); i++) {
      MimeBodyPart bp = (MimeBodyPart)mp.getBodyPart(i);
      Object o = p.getContent();
   } (...)
```

Envio de mensagens com anexos

- *Para enviar anexos enviamos a mensagem como Multipart*
  - *Objeto MimeMultipart serve de container para tipos MIME diferentes*
  - *Para texto da mensagem e cada anexo é preciso criar um objeto MimeBodyPart e preenche-lo através do método setContent()*
  - *No final, coloca-se o MimeMultipart em uma mensagem e envia-se*

```
Message msg = new MimeMessage(session);
(...)
Multipart mp = new MimeMultipart();

MimeBodyPart mbp1 = new MimeBodyPart();
mbp1.setContent(args.length + " files attached.", "text/plain");
mp.addBodyPart(mbp1);

for (int i = 0; i < args.length; i++) {
    File f = new File(args[i]);
    MimeBodyPart mbp = new MimeBodyPart();
    mbp.setFileName (f.getName());
    mbp.setDataHandler(new DataHandler(new FileDataSource(f)));
    mp.addBodyPart(mbp);
}
msg.setContent(mp);
Transport.send(msg);
```

Execução dos exemplos

- *Configuração do ambiente*
  - *Primeiro configure o arquivo smtp.properties (em lib/) e informe o endereço do servidor de e-mail utilizado, nome e senha.*
  - *Se quiser usar um servidor local, você pode usar o Jakarta James que suporta POP, NNTP e SMTP. Inicie o James.*
- *Execução (em cap17/nut/)*
  - *Envio de mensagens simples*

```
> ant sendmail
```

  - *Envio de mensagens com attachment: passe a lista de arquivos entre aspas para o argumento "-Dattach" no build.xml*

```
> ant sendmail -Dattach="java.gif build.xml"
```

  - *Listagem das mensagens no INBOX*

```
> ant getmail
```

  - *Para ver o conteúdo (e possívelmente esvaziar o INBOX) use seu cliente de e-mail (ou escreva um que faça isto!)*

Conclusões

- *JavaMail oferece uma API que permite transformar qualquer aplicação Java em um cliente IMAP, SMTP, POP3 ou NNTP*
- *Permite capacitar servlets, JSPs, EJBs e outros componentes com recursos de e-mail*
  - *Pode ser usado para messaging XML (Web Services)*
  - *Pode ser usado para transferir arquivos de forma assíncrona (usando attachments)*

Exercícios

- 1. Escreva uma página JSP simples que ofereça uma interface para
  - Escrever uma mensagem
  - Fazer upload de um arquivo (attachment). Use a biblioteca cos.jar em cap06\hunter\upload\lib (veja exemplos na mesma pasta)
  - Enviar mensagem com arquivo atachado
- 2. Escreva uma página JSP que
  - Conecte-se ao servidor de e-mail
  - Liste as mensagens disponíveis: Data de envio, remetente, assunto e link para attachments.

Fontes

[1] *Jim Farley, William Crawford e David Flanagan. Java Enterprise in a Nutshell.2nd. Edition, 2002. O'Reilly.*

helder@ibpinet.net

www.argonavis.com.br

J500: Java 2 Enterprise Edition

Helder da Rocha
www.argonavis.com.br

Sumário

- *Parte I: XML e tecnologias relacionadas*
  - *Documentos XML; namespaces; processadores*
  - *DTD e XSchema*
  - *DOM e SAX*
  - *XSLT e XPath*
- *Parte II: APIs Java para XML e Web Services*
  - *Introdução a Web Services: SOAP, WSDL, UDDI*
  - *JAXP - Java API for XML Processing, JDOM, SAX e XSLT*
  - *JAXB - Java API for XML Binding*
  - *JAXM - Java API for XML Messaging*
  - *JAXR - Java API for XML Registry*
  - *JAXRPC - como criar um Web Service com JAX-RPC*
  - *Java Web Service Development Kit*

Parte I
Introdução a XML e tecnologias relacionadas

Parte I - Introdução a XML

Objetivos

- *Responder às questões*
  - ***Como implementar** soluções de gestão de informações usando XML?*
  - ***Quando e como usar** as tecnologias e linguagens que viabilizam o compartilhamento de informações?*
- *Apresentar*
  - *Breve **introdução ao XML** e tecnologias relacionadas.*
  - *Recursos para manipular informações representadas em XML: **ferramentas, linguagens e tecnologias***

Parte I - Programa

Por que XML? Onde usar XML?
Como produzir documentos XML
Documentos válidos: DTD e XML Schema

Manipulação via programação em DOM e SAX
Transformação: XSLT e XPath
Localização e extração: XLink, XQuery e XPointer
Visualização: XSL-FO e SVG

Demonstração: geração de HTML, RTF e PDF

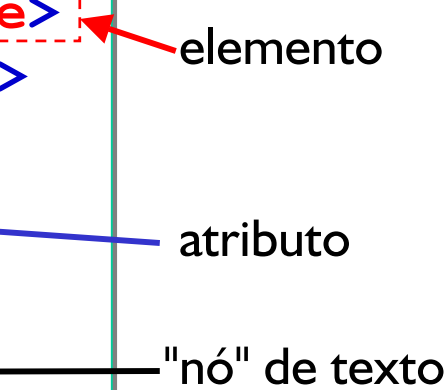
Ferramentas e Conclusões

O que é XML?

- e**X**tensible **M**arkup **L**anguage: padrão W3C
- Uma maneira de **representar** informação
  - não é uma linguagem específica
  - não define vocabulário de comandos
  - não define uma gramática, apenas regras mínimas
- Exemplo: documento XML

usuario_33.xml

```
<contato codigo="33">
  <nome>Severino Severovitch</nome>
  <email>bill@norte.com.br</email>
  <telefone tipo="celular">
    <area>11</area>
    <numero>9999 4321</numero>
  </telefone>
</contato>
```



XML versus HTML

HTML mostra ***como*** *apresentar*

```
<h1>Severino Severovitch</h1>
<h2>bill@norte.com.br</h2>
<p>
 <b>11</b>
 <i>9999 4321</i>
</p>
```

XML mostra ***o que*** *significa*

```
<nome>Severino Severovitch</nome>
<email>bill@norte.com.br</email>
<telefone>
 <ddd>11</ddd>
 <numero>9999 4321</numero>
</telefone>
```

Anatomia de um documento XML

- *Documentos XML são documentos de texto Unicode*
  - *É uma hierarquia de **elementos** a partir de uma **raiz***
  - *Menor documento tem um elemento (vazio ou não):*

```
<nome> Северино Северович </nome>
```

Elemento raiz

  - *Menor documento contendo elemento vazio*

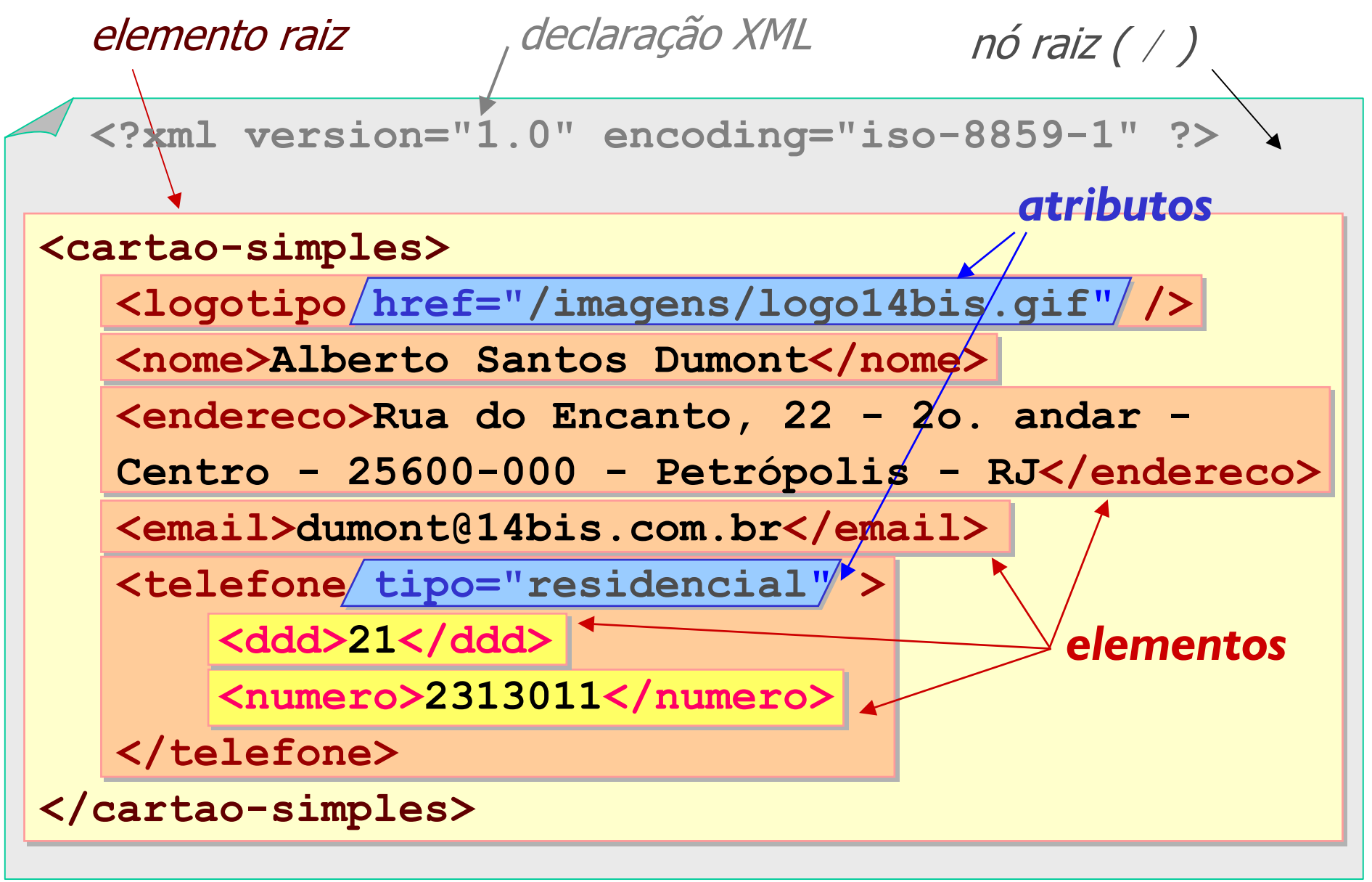
```
<nome></nome>   =   <nome/>
```

  - *Menor documento contendo elemento e conteúdo texto*

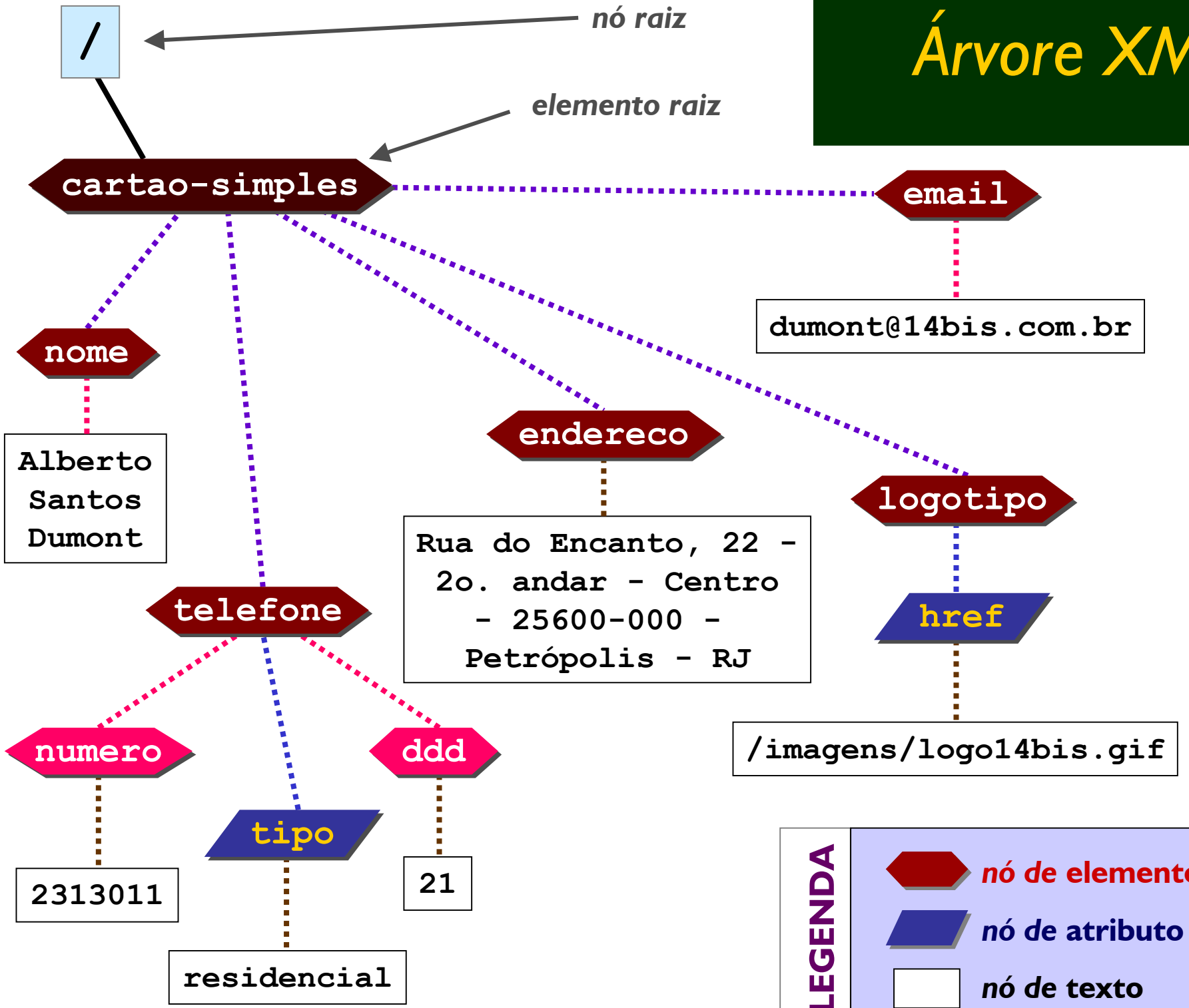
```
<nome> Северино Северович </nome>
```

Etiqueta inicial — *Conteúdo do Elemento* — *Etiqueta final*

Partes de um documento

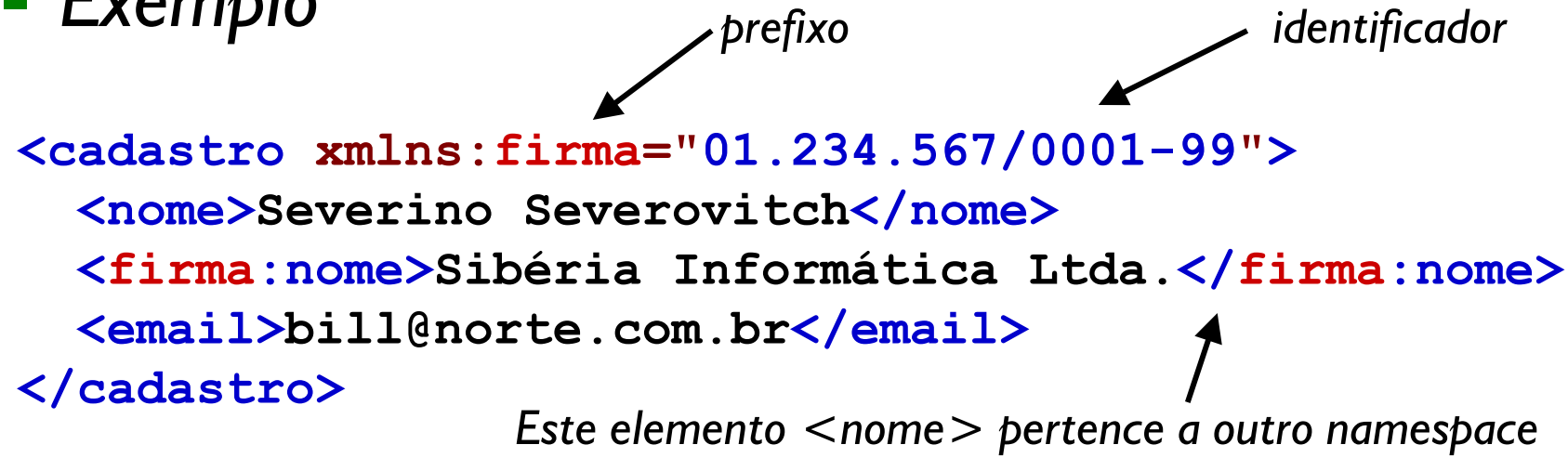


Árvore XML


/
nó raiz
elemento raiz
cartao-simples
email
dumont@14bis.com.br
nome
Alberto Santos Dumont
endereco
Rua do Encanto, 22 - 2o. andar - Centro - 25600-000 - Petrópolis - RJ
logotipo
href
/imagens/logo14bis.gif
telefone
numero
2313011
tipo
residencial
ddd
21
LEGENDA
nó de elemento
nó de atributo
nó de texto

XML Namespaces

- *Limita o escopo de elementos*
  - *Evita conflitos quando duas linguagens se cruzam no mesmo documento*
- *Consiste da associação de um **identificador** a cada elemento/atributo da linguagem, que pode ser*
  - *herdado através do escopo de uma sub-árvore*
  - *atribuído explicitamente através de um prefixo*
- *Exemplo*

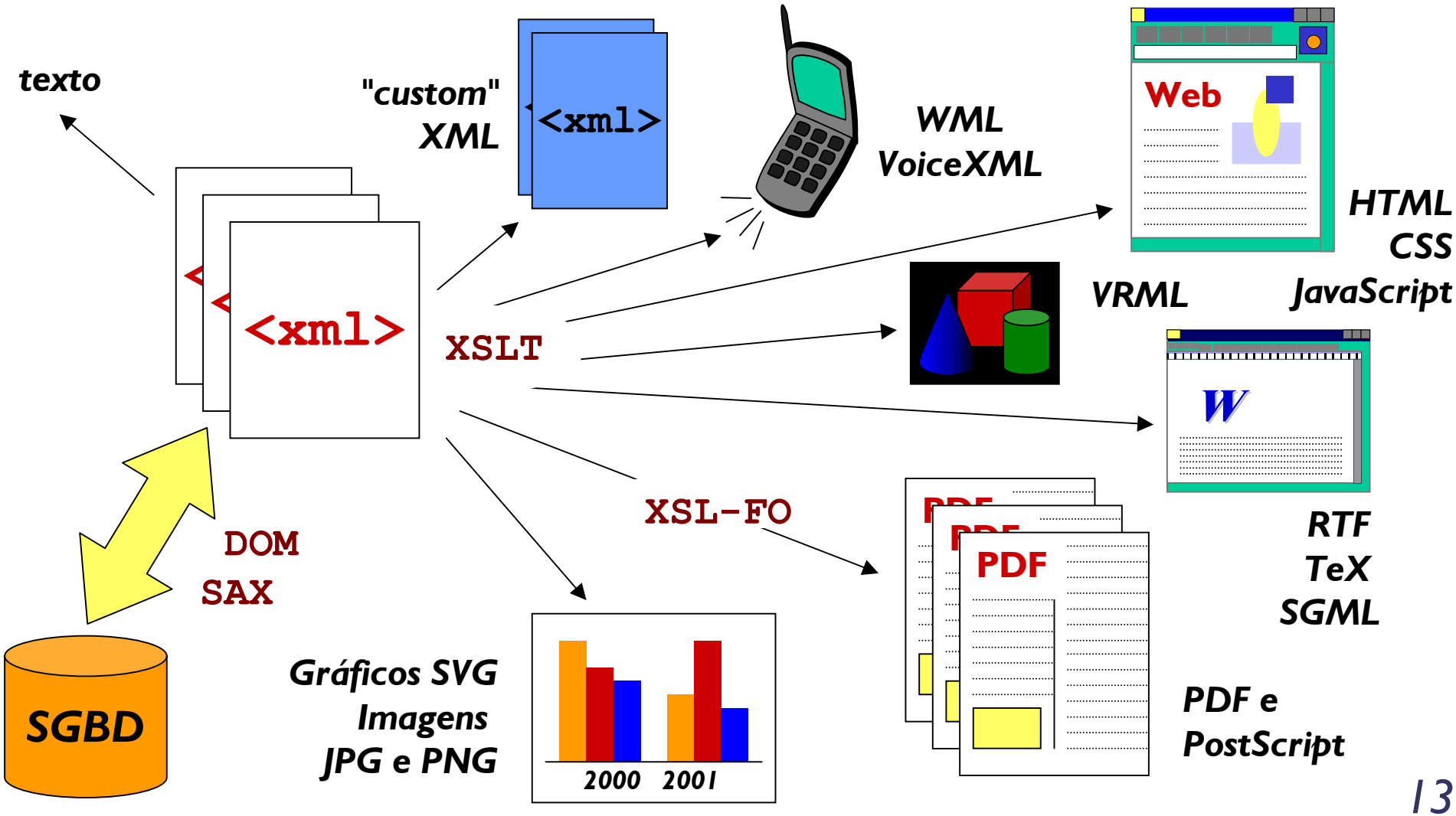


Por que usar XML para compartilhar dados?

- Porque é um **padrão aberto**
  - Facilidade para converter para formatos proprietários
- Porque é **texto**
  - Fácil de ler, fácil de processar, menos incompatibilidades
- Porque promove a **separação** entre estrutura, conteúdo e apresentação
  - Facilita geração de dados para visualização dinâmica
  - Evita repetição de informação / simplifica manutenção
- Porque permitirá **semântica** na Web
  - Elementos HTML não carregam significado, apenas dicas de formatação: mecanismos de busca ficam prejudicados
  - Solução com XML dependerá de suporte dos clientes

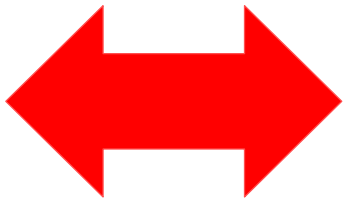
Onde usar XML?

- *Dados armazenados em XML podem ser facilmente transformados em outros formatos*



Como produzir XML

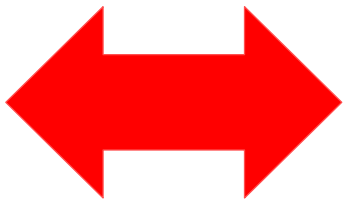
- ***Criando*** *um documento de texto Unicode a partir de qualquer editor de textos*



```
<contato codigo="33">
  <nome>Severino Severovitch</nome>
  <email>bill@norte.com.br</email>
  <telefone tipo="celular">
    <area>11</area>
    <numero>9999 4321</numero>
  </telefone>
</contato>
```

- ***Gerando*** *um documento a partir de uma árvore montada dinamicamente*

- *33*
- *Severino Severovitch*
- *bill@norte.com.br*
- *celular*
- *11*
- *9999 4321*



```
<contato codigo="33">
  <nome>Severino Severovitch</nome>
  <email>bill@norte.com.br</email>
  <telefone tipo="celular">
    <area>11</area>
    <numero>9999 4321</numero>
  </telefone>
</contato>
```

Documentos XML bem formados

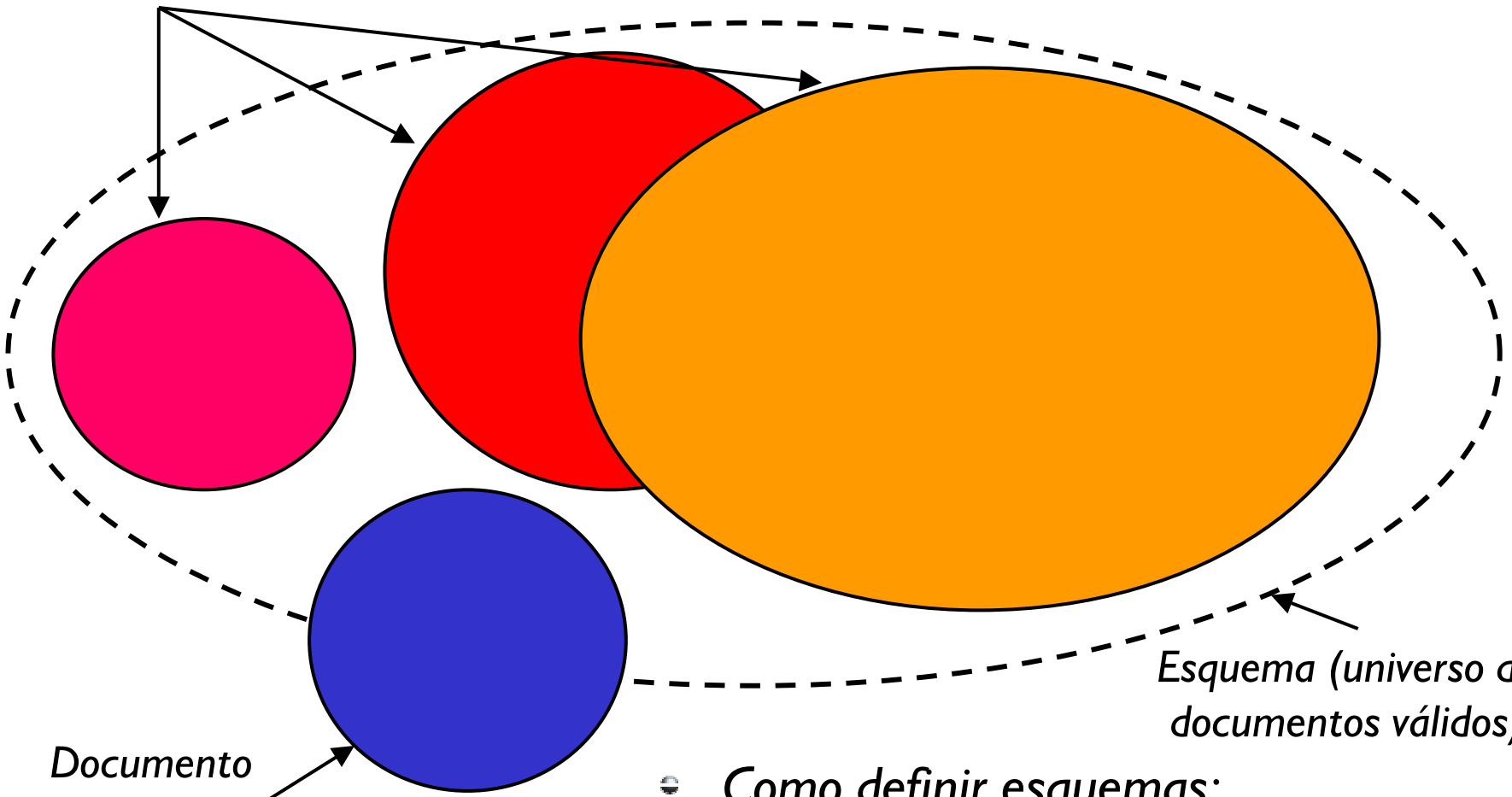
- Para que possa ser manipulado como uma árvore, um documento XML precisa ser **bem formado**
  - Documentos que não são bem formados não são documentos XML
- Documentos bem-formados obedecem as regras de construção de documentos XML genéricos
- Regras incluem
  - Ter um, e apenas um, elemento raiz
  - Valores dos atributos estarem entre aspas ou apóstrofes
  - Atributos não se repetirem
  - Todos os elementos terem etiqueta de fechamento
  - Elementos estarem corretamente aninhados

XML válido

- *Um XML bem construído pode não ser **válido** em determinada aplicação*
- *Aplicação típica pode esperar que*
  - *elementos façam parte de um **vocabulário** limitado,*
  - *certos atributos tenham **valores** e **tipos** definidos,*
  - *elementos sejam organizados de acordo com uma determinada estrutura **hierárquica**, etc.*
- *É preciso **especificar** a linguagem!*
  - ***Esquema**: modelo que descreve todos os elementos, atributos, entidades, suas relações e tipos de dados*
- *Um documento XML é considerado válido **em relação a um esquema** se obedecer todas as suas regras*

Esquema

- *O esquema representa uma **classe***
- *Os documentos são **instâncias***



- *Como definir esquemas:*
  - ***DTD** - Document Type Definition*
  - ***W3C XML Schema***

DTD vs. XML Schema

- *Um esquema é essencial para que haja* ***comunicação*** *usando XML*
  - *Pode ser estabelecido "informalmente" (via software)*
  - *Uso formal permite validação usando ferramentas genéricas de manipulação de XML*
- *Soluções*

DTD

```
<!ELEMENT contato
          (nome, email, telefone)>
<!ATTLIST contato
          codigo NMTOKEN #REQUIRED>
```

- *Simples mas não é XML*
- *Não suporta namespaces*
- *Limitado quando a tipos de dados*

XSchema

```
<xsd:schema
      xmlns:xsd=".../XMLSchema">
 <xsd:element name="contato">
  <xsd:complexType>
   <xsd:attribute name="codigo"
                  use="required">
```

- *É XML, porém mais complexo*
- *Suporta namespaces*
- *Permite definição de tipos*

Visualização em um browser

- *Folha de estilo*: conjunto de regras para formatar ou transformar as informações de um documento XML
- ***CSS*** - Cascading Style Sheets
  - Transformação visando apresentação visual
  - Aplicação do estilo em tempo de execução no cliente
- ***XSLT*** - eXtensible Stylesheet Language
  - Transformação em texto, HTML ou outro formato
  - Aplicação em tempo real ou prévia (no servidor)
- Se não estiver associado a uma folha de estilo, o documento XML não tem uma "aparência" definida
  - Internet Explorer e outros mostram a árvore-fonte XML
  - Netscape mostra apenas os nós de texto

Como manipular XML?

- Há duas APIs padrão para manipular (interpretar, gerar, extrair dados e tratar eventos) arquivos XML:
  - W3C Document Object Model (W3C DOM)
  - Simple API for XML (SAX)
- Servem a finalidades diferentes
- Implementações disponíveis em várias linguagens
- **SAX** oferece métodos que respondem a **eventos** produzidos durante a leitura do documento
  - notifica quando um elemento abre, quando fecha, etc.
- **DOM** monta uma **árvore**, que permite a navegação na estrutura do documento
  - propriedades dos objetos podem ser manipuladas

Leitura de XML com SAX

- *Se um processador SAX receber o documento ...*

```
<carta>
  <mensagem id="1">Bom dia!</mensagem>
</carta>
```

- *... ele irá disparar a seguinte seqüência de eventos:*

```
startDocument()
  startElement("carta", [])
    startElement("mensagem", [Attribute("id","1")])
      characters("Bom dia!")
    endElement("mensagem")
  endElement("carta")
endDocument()
```

- *Programador deve implementar um objeto "ouvinte" para capturar os eventos e extrair as informações desejadas*

Criação de documentos com DOM (I)

- *Criação dos elementos*

/	**Document**	*Obter objeto do tipo* ***Document*** *(raiz)* *(dependente de processador):* `doc`
<carta>	**Element**	`carta = doc.createElement("carta")`
<mensagem>	**Element**	`mens = doc.createElement("mensagem")`
Bom dia!	**String**	`texto = doc.createTextNode("Bom dia!")`

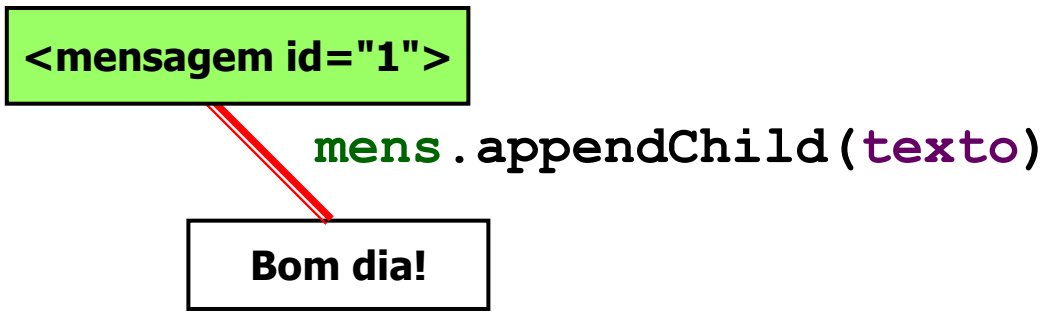
- *Atributos*

<mensagem id="1">	`mens.setAttribute("id", "1")`

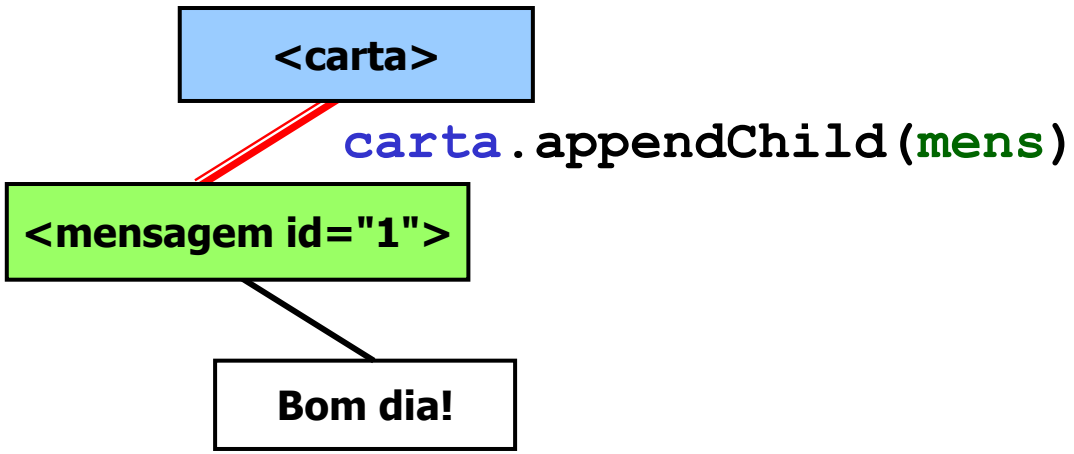
Criação de documentos com DOM (2)

Montagem da árvore passo-a-passo

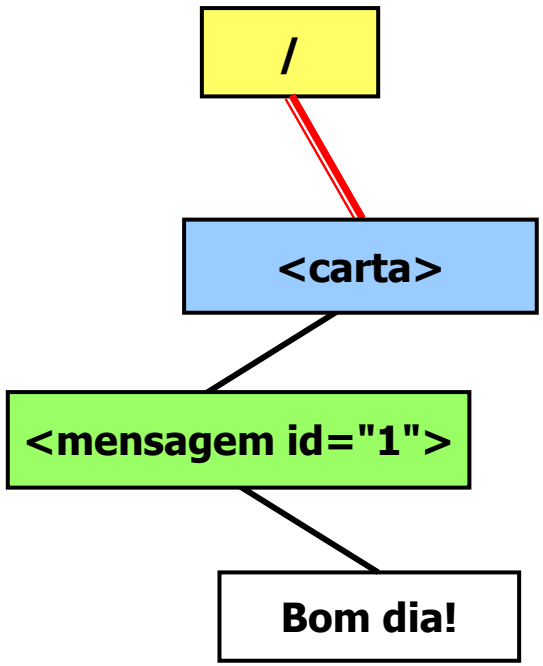
1. Sub-árvore <mensagem>



2. Sub-árvore <carta>



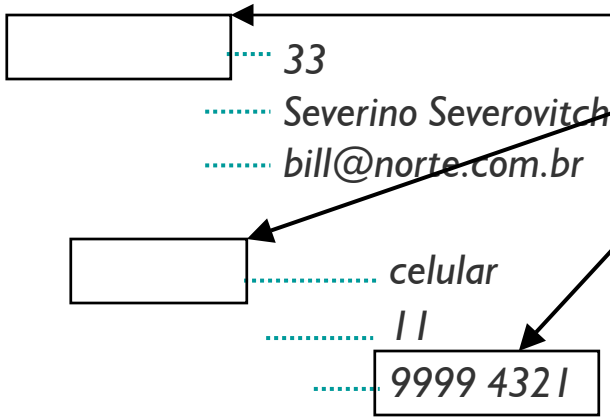
3. Árvore completa




```
doc.appendChild(carta)
```

XPath

- *Linguagem usada para **navegar** na árvore XML*
  - *Uma **expressão XPath** é um caminho* na árvore que resulta em um valor (número, texto, booleano), objeto (elemento, atributo, nó de texto) ou conjunto de objetos*



- *Caminhos absolutos*

```
/contato/@codigo
/contato/telefone/@tipo
/contato/telefone/numero/text()
```

- *Relativos ao contexto* `/contato`:

```
@codigo (ou ./@codigo)
telefone@tipo (ou ./telefone/tipo)
telefone/numero/text()
```

- *Expressões XPath são usadas dentro de **atributos** XML*
  - *Usadas em XSLT, XLink, XQuery e XPointer*

* pode também ser padrão de busca

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="livro/titulo">
      <td><xsl:value-of select="." /></td>
```

XSLT

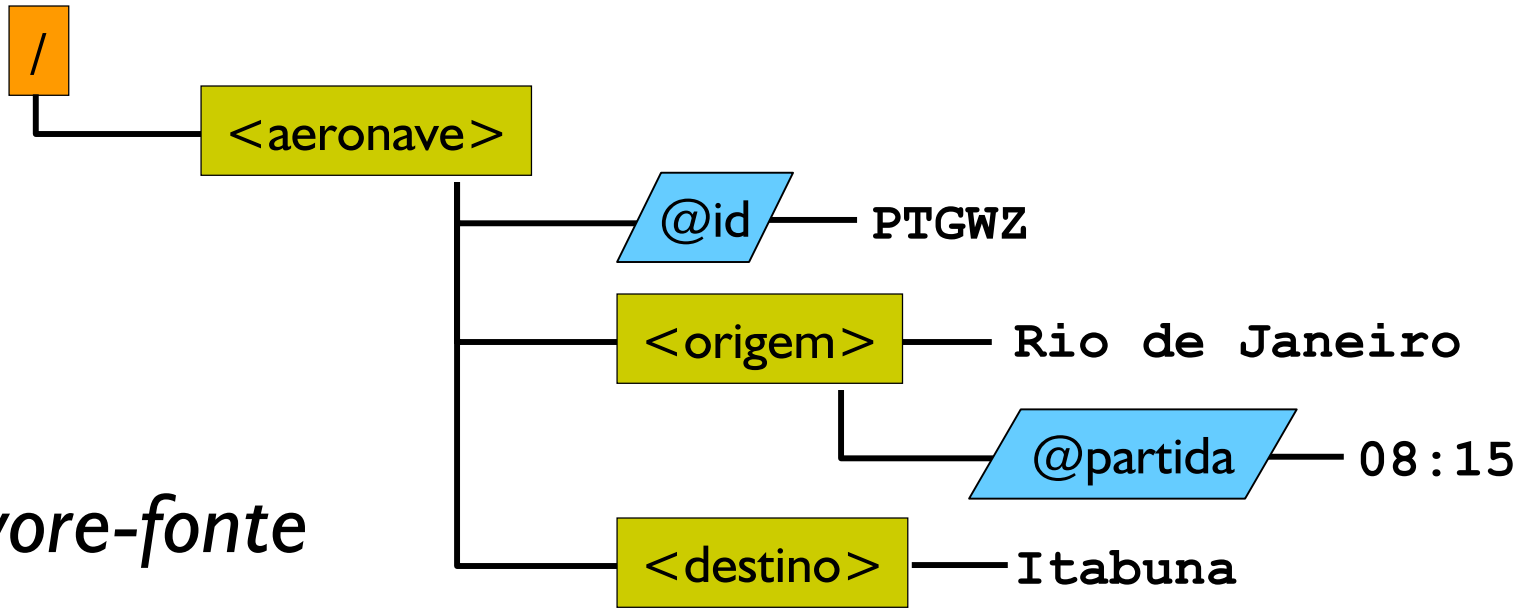
- ***XSL Transformations***
  - *Linguagem (XML) para criação de documentos que contêm regras de transformação para documentos XML*
  - *Documentos escritos em XSLT são chamados de* ***folhas de estilo*** *e contêm*
    - ***Elementos XSLT****: <template>, <if>, <foreach>, ...*
    - ***Expressões XPath*** *para localizar nós da árvore-fonte*
    - ***Texto ou XML*** *a ser gerado no documento-resultado*
  - *Processador XSLT*

XSLT: documento-fonte (I)

- *Considere o seguinte documento-fonte:*

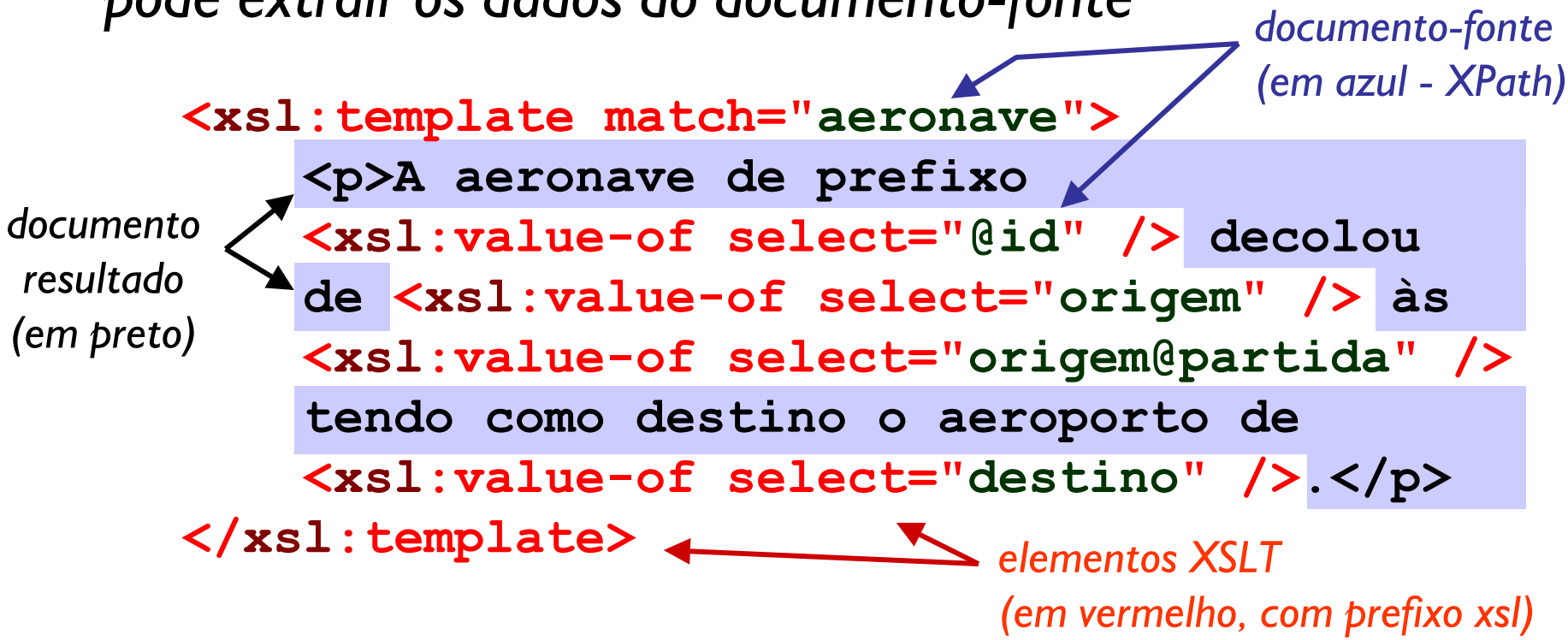
```
<aeronave id="PTGWZ">
   <origem partida="08:15">Rio de
                                Janeiro</origem>
   <destino>Itabuna</destino>
</aeronave>
```

- *Árvore-fonte*



XSLT: folha de estilos (2)

- *O seguinte **template** (parte de uma folha de estilos XSLT) pode extrair os dados do documento-fonte*



- *Elementos XSLT geralmente são usados com um **prefixo** associado ao seu **namespace**: <xsl:elemento> para evitar conflitos com o documento-resultado.*

XSLT: documento-resultado (3)

- Após a transformação, o resultado será

```
<p>A aeronave de prefixo
PTGWZ decolou
de Rio de Janeiro às
8:15
tendo como destino o aeroporto de
Itabuna.</p>
```

- Para obter outros resultados e gerar outros formatos com os mesmos dados, deve-se criar folhas de estilo adicionais

XLink, XPointer e XQuery

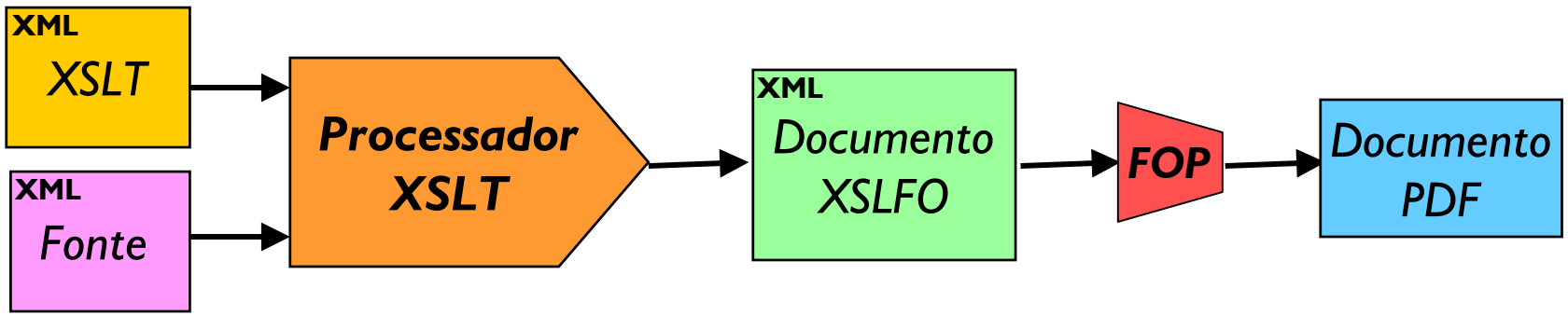
- ***XLink*****:** *é uma especificação W3C que permite definir vínculos entre documentos XML*
  - *Funcionalidade mínima é igual ao <a href> do HTML*
  - *Funcionalidade estendida permite vínculos bidirecionais, arcos, vários níveis de semântica, etc.*
  - *É uma coleção de atributos, com namespace próprio, que podem ser usados em elementos de qualquer linguagem XML.*
- ***XPointer*****:** *aponta para partes de documentos XML*
  - *Identificador (ID) colocado no destino, acessível através de fragmento de URL:* `xlink:href="#identificador"`
  - *Caminho resultante de expressão XPath:* `xpointer(/livro/id)`
- ***XQuery*****:** *linguagem para pesquisar documentos XML*
  - *Exemplo:*

```
FOR $b IN document("usuario_33.xml")/contato
WHERE nome="Severino Severovitch"
RETURN $b
```

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <fo:layout-master-set>
      <fo:simple-page-master master-name="p1">
         <fo:region-body/>
```

XSL-FO

- ***XSL Formatting Objects***
  - *Linguagem XML de **descrição de página** com os mesmos recursos que PostScript ou PDF*
  - *Descreve o layout preciso de texto e imagens*
  - *Possui centenas de elementos, atributos e propriedades (que são semelhantes às propriedades do CSS)*
  - *Páginas são facilmente convertidas para PDF e PostScript*
  - *Ideal para gerar documentos para impressão (livros, etc.)*
- *Normalmente **gerada** via XSLT*



XSL-FO: menor documento

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

   <fo:layout-master-set>
       <fo:simple-page-master master-name="p1">
           <fo:region-body/>
       </fo:simple-page-master>
   </fo:layout-master-set>

   <fo:page-sequence master-name="p1">
       <fo:flow flow-name="xsl-region-body">
          <fo:block color="blue" font-size="20pt">
             Hello PDF!
          </fo:block>
       </fo:flow>
   </fo:page-sequence>

</fo:root>
```

Este é o "<head>" do XSL-FO

Ligação entre as regras de layout e o conteúdo afetado

Este é o "<body>" do XSL-FO

```
<html xmlns="http://www.w3.org/1999/xhtml">
   <head><title>Página XHTML</title></head>
   <body>
      <h1>Página XHTML</h1>
```

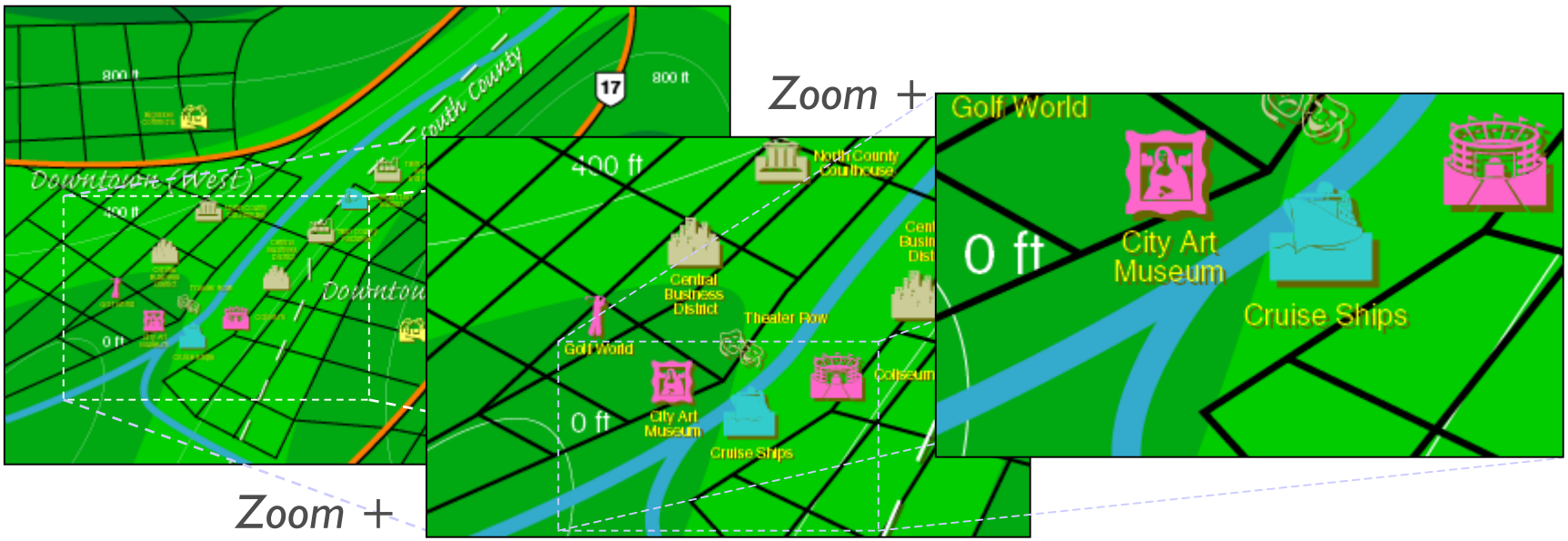
XHTML

- e**X**tensible **HTML**
  - *Linguagem XML de* ***descrição de página Web***
  - *Mesmos elementos do HTML 4.0 Strict*
  - *Elementos descrevem somente a estrutura dos componentes da página.*
    - *A forma precisa ser especificada usando CSS: não há elementos/atributos para mudar cor, alinhamento, etc.*
  - *Pode ser misturada (estendida) com outras linguagens XML (MathML, SVG, linguagens proprietárias)*
- *Normalmente* ***gerada*** *via XSLT*

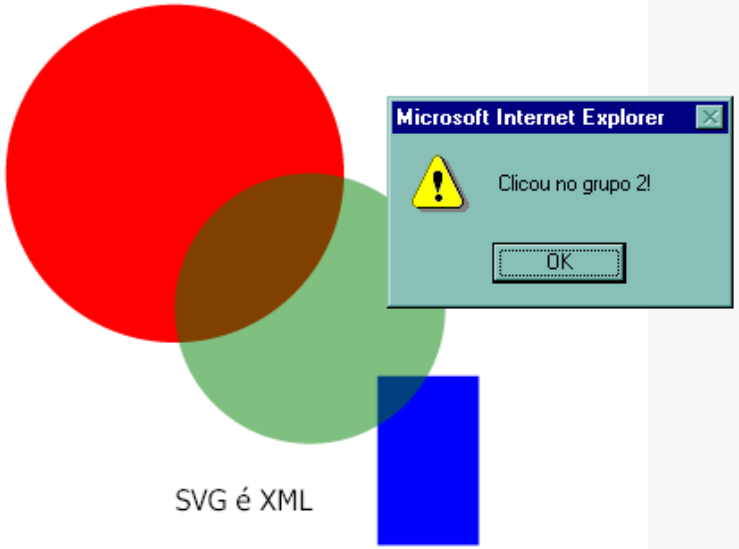
```
<svg>
   <circle style="fill: red" cx="3cm" cy="3cm" r="2.5cm" />
   <rect style="fill: blue" x="6cm" y="6cm"
         height="2.5cm" width="1.5cm" />
```

SVG

- *W3C **S**calable **V**ector **G**raphics*
  - *Gráficos vetoriais em XML*
  - *Plug-ins para principais browsers: concorre com Flash*
  - *Suporta animações, links, JavaScript, CSS*
  - *Produzido por ferramentas como Adobe Illustrator*
  - *Pode ser embutido no código XHTML e XSL-FO*



Exemplo de SVG



JavaScript

CSS

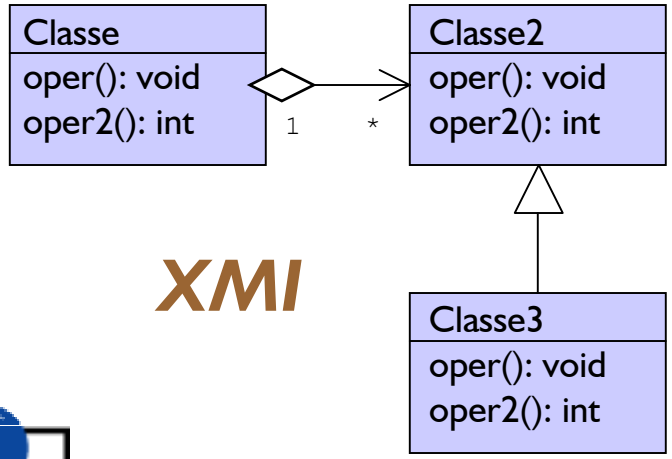
XLink

```
<svg width="10cm" height="10cm">
  <g onclick="alert('Clicou no grupo 1!')">
    <circle style="fill: red"
            cx="3cm" cy="3cm" r="2.5cm" />
    <rect style="fill: blue" x="6cm" y="6cm"
          height="2.5cm" width="1.5cm" /></g>
  <g onclick="alert('Clicou no grupo 2!')">
    <circle style="fill: green; opacity: 0.5"
            cx="5cm" cy="5cm" r="2cm" /></g>
  <a xmlns:xlink="http://www.w3.org/1999/xlink"
     xlink:href="http://www.w3.org/Graphics/SVG">
    <text style="color: black; font-family: tahoma;
                 font-size: 12pt" x="3cm" y="8cm">
    SVG é XML</text></a>
</svg>
```

Algumas outras linguagens XML

MathML

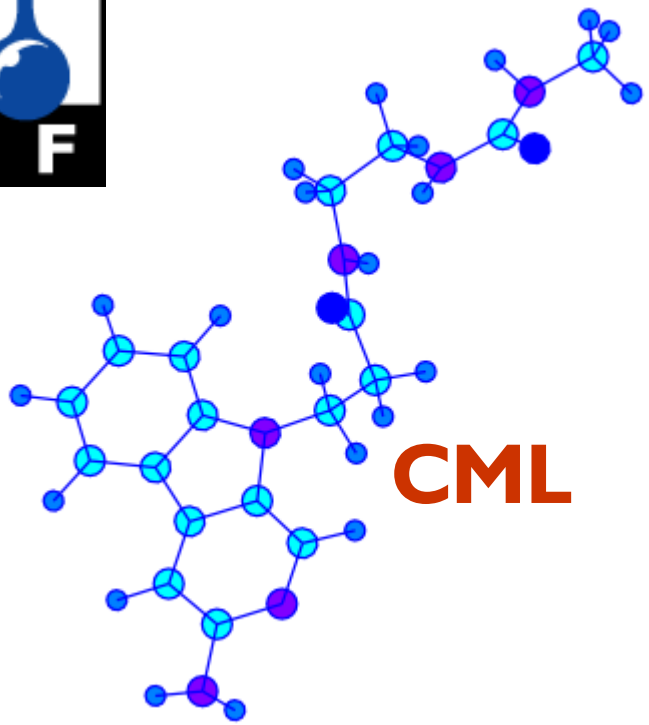
WML
VoiceXML



XMI

Web Services

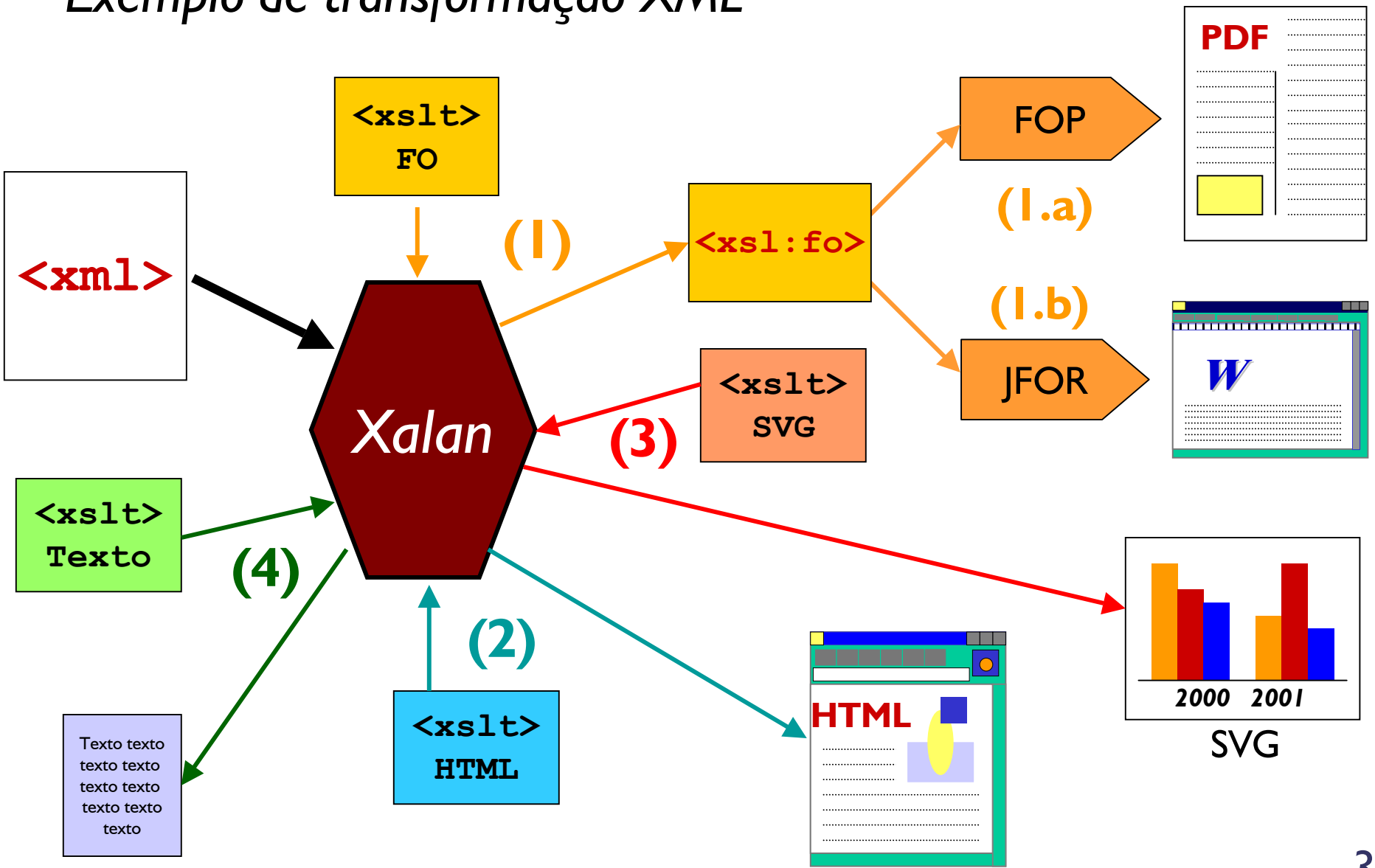
SOAP
WSDL
UDDI
XML-RPC
ebXML



CML

Demonstração

- *Exemplo de transformação XML*



Ferramentas e APIs Java

- Para programação
  - Parsers-validadores: Xerces, Crimson, MSXML 4.0
  - Validadores: MSV (Sun)
  - Transformadores XSL: TrAX (JAXP), Xalan, Xt, Saxon
  - APIs: JDOM, JAXP (DOM, SAX)
  - Veja mais em xml.apache.org e www.alphaworks.ibm.com
- Para edição (de XML genérico)
  - XML Spy Suite
  - Framemaker / ArborText
  - JEdit com plug-ins para XML, XSLT e XPath
  - Veja mais em www.w3.org/XML/

Conclusões

- *XML é uma ótima solução para **compartilhar** dados*
- *Para **implementar** soluções em gestão de informações usando XML, pode-se usar*
  - *DTD ou XSchema para especificar o modelo de dados e validar as informações*
  - *As APIs DOM ou SAX para extrair dados dos documentos, gerar documentos, ler e gravar em bancos de dados*
  - *XSLT e XPath para transformar os dados em outros formatos*
  - *XLink, XPointer e XQuery para criar vínculos lógicos entre os documentos e localizar seus componentes*
  - *XSL-FO ou XHTML para formatar os dados para impressão ou visualização na tela (PDF, Word ou Web)*
  - *SVG para gerar informações em forma de gráfico vetorial*

Fontes

[1] World Wide Web Consortium (W3C). eXtensible Markup Language. http://www.w3.org/XML/. Ponto de partida e principal fonte sobre XML e suas tecnologias "satélite". Contém últimas especificações de XML, XPath, XSchema, XSLT, XSL-FO, XQuery, XLink, XPointer, SVG, XHTML, CSS.

[2] Eric Armstrong et al. Working with XML. Aborda DOM, SAX e XML com Java. http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/index.html.

[3] Adobe. SVG Tutorial. http://www.adobe.com/svg/. Contém tutorial sobre SVG e links para o plug-in SVG da Adobe (Win/Mac).

[4] IBM Developerworks. http://www-106.ibm.com/developerworks/. Diversos tutoriais e artigos sobre XML, XSLT, DOM e SAX usando geralmente Java.

[5] Doug Tidwell. XSLT. O'Reilly & Associates, 2001. Explora XSLT com aplicações práticas em Java.

[6] Elliotte Rusty Harold. XML Bible, Second Edition, 2001. Aborda todas as principais tecnologias W3C. 5 capítulos em http://cafeconleche.org/books/bible2/

[7] Erik T. Ray. Learning XML. O'Reilly & Associates, 2001. Introdução ao XML e DTD, XSLT, XLink e XPointer (os dois últimos baseados em especificações draft).

Parte II
Java, XML e Web Services

Parte II - Java, XML e Web Services

- *Objetivos:*
  - *Definir Web Services*
  - *Descrever as tecnologias XML padrão que oferecem suporte a Web Services*
  - *Descrever as APIs Java distribuídas com o Java Web Services Development Pack 1.0*
- *Mostrar como criar um Web Service*
  - *Utilizar a API JAX-RPC para desenvolver e implantar um Web Service simples baseado no protocolo SOAP*
  - *Gerar uma interface WSDL e utilizá-la para construir um cliente para o serviço*
  - *Registrar uma organização e a localização do arquivo WSDL em um servidor UDDI local*

O que são Web Services

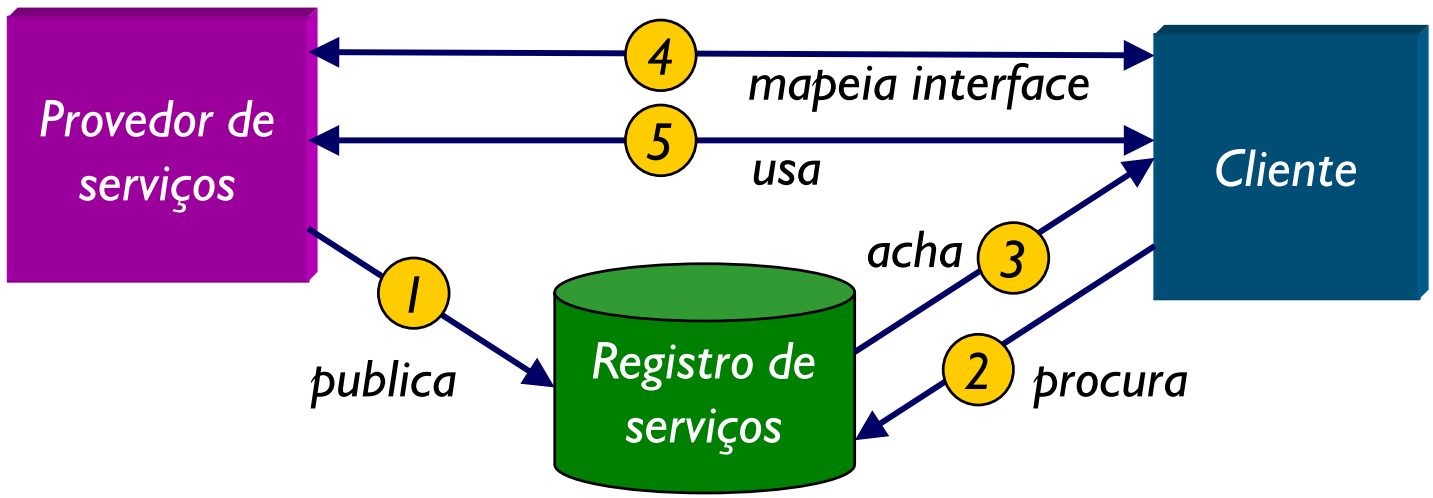
- Ambiente de computação distribuída (DCE) que utiliza XML em todas as camadas
  - No formato de dados usado na comunicação
  - Na interface usada para descrever as operações suportadas
  - Na aplicação usada para registrar e localizar serviços
- Serviços são transportados principalmente via HTTP
  - Podem também utilizar outros protocolos populares
- Web Services visam comunicação entre máquinas
  - Serviços podem ser implementados usando CGI (com C, Perl, etc.), ASP, PHP, servlets, JSP, CFML, etc.
  - Acesso é feito via clientes HTTP (ou de outros protocolos)
- Tudo isto já existia! Qual a novidade?

A novidade é a padronização!

- Todas as camadas em XML!
  - Fácil de ler, transformar, converter
  - Existe ainda um esforço para padronizar os esquemas que definem a estrutura e vocabulário do XML usado
- Web Services dá nova vida ao RPC
  - Agora com formato universal para os dados!
  - ➡ Marshalling: converter dados em XML
  - ➡ Unmarshalling: extrair dados de XML
- Principais características do RPC com Web Services
  - Formato padrão de dados usados na comunicação é XML
  - Interoperabilidade em todos os níveis
  - Transporte é protocolo de larga aceitação: HTTP, SMTP,...
  - Transparência de localidade e neutralidade de linguagem

Arquitetura de Web Services: papéis

- Provedor de serviços
  - Oferece serviços, alguns dos quais podem ser Web Services
- Registro de serviços
  - Catálogo de endereços: repositório central que contém informações sobre web services
- Cliente de serviços
  - Aplicação que descobre um web service, implementa sua interface de comunicação e usa o serviço

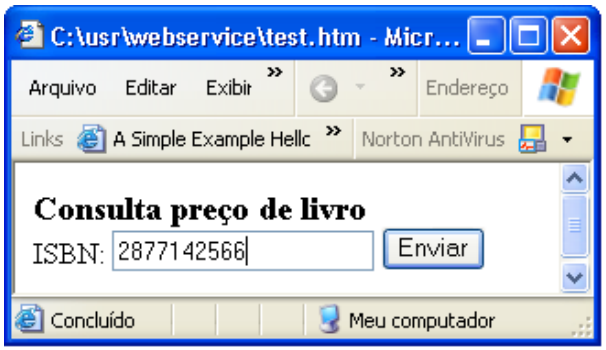


Arquitetura de Web Services: camadas

- Camada de transporte
  - Principais: HTTP (POST), FTP, SMTP
  - Emergentes: JRMP (Java RMI), IIOP (CORBA, EJB), JMS, IMAP, POP, BEEP, JXTA, ...
- Camada de mensagens
  - SOAP
- Camada dados ou serviços
  - XML (formato de mensagens)
  - XML-RPC
- Camada de descrição de serviços
  - WSDL
- Camada de descoberta (registro)
  - UDDI, ebXML

Requisição e resposta HTTP POST

- Clientes HTTP usam o método POST para enviar dados
  - Tipicamente usado por browsers para enviar dados de formulários HTML e fazer upload de arquivos
- Exemplo
  - Formulário HTML




```
<FORM ACTION="/cgi-bin/catalogo.pl"
      METHOD="POST">
 <H3>Consulta preço de livro</H3>
 <P>ISBN: <INPUT TYPE="text" NAME="isbn">
 <INPUT TYPE="Submit" VALUE="Enviar">
</FORM>
```

  - Requisição POST gerada pelo browser para o servidor

Cabeçalho HTTP

Linha em branco

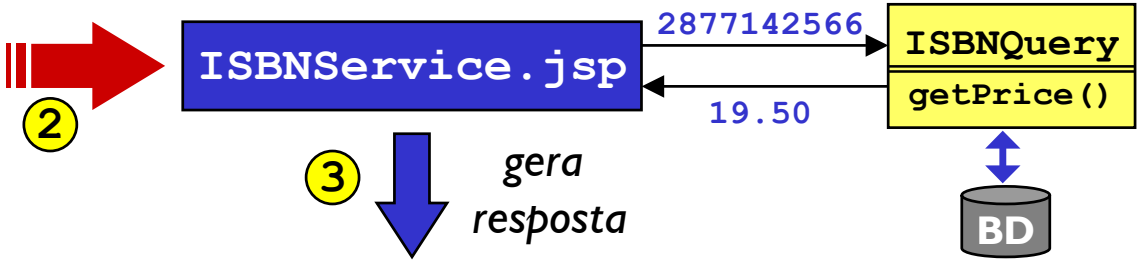
Mensagem (corpo da requisição)

```
POST /cgi-bin/catalogo.pl HTTP/1.0
Content-type: text/x-www-form-urlencoded
Content-length: 15

isbn=2877142566
```

Enviando XML sobre POST

- Você pode criar um servico RPC simples (um Web Service!) trocando mensagens XML via HTTP POST!
  - Defina esquemas para as mensagens de chamada e resposta
  - Escreva cliente que envie requisições POST para servidor Web
  - Escreva uma aplicação Web (JSP, ASP, PHP, servlet, CGI)

```
POST /ISBNService.jsp HTTP/1.0
Content-type: text/xml
Content-length: 90

<chamada>
  <funcao>
    <nome>getPrice</nome>
    <param>2877142566</param>
  </funcao>
</chamada>
```




```
HTTP/1.1 200 OK
Content-type: text/xml
Content-length: 77

<resposta>
  <funcao>
    <param>19.50</param>
  </funcao>
</resposta>
```

XML-RPC

- *Especificação para RPC em XML via HTTP POST*
  - *Projetada para ser a solução mais simples possível*
  - *Várias implementações: veja www.xml-rpc.com*
- *Exemplo anterior implementado com XML-RPC (cabeçalhos HTTP omitidos)*

```
<methodCall>
  <methodName>getPrice</methodName>
  <params>
    <param>
      <value><string>2877142566</string></value>
    </param>
  </param>
</methodCall>
```

Requisição

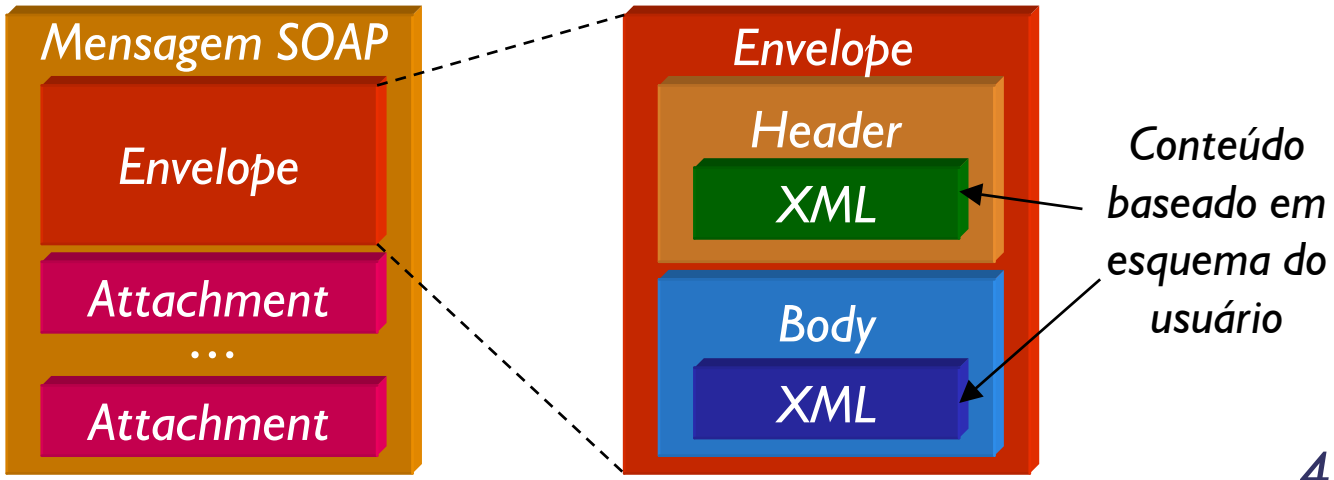
Resposta

```
<methodResponse>
    <params>
    <param>
      <value><double>19.5</double></value>
    </param>
  </param>
</methodResponse>
```

SOAP

- **S**imple **O**bject **A**ccess **P**rotocol
- Protocolo padrão baseado em XML para trocar mensagens entre aplicações
  - SOAP não é um protocolo RPC, mas um par de mensagens SOAP pode ser usado para esse fim
  - Transporte pode ser HTTP, SMTP ou outro
  - Mensagens podem conter qualquer coisa (texto, bytes)
  - É extensível (mecanismo de RPC, por exemplo, é extensão)

Estrutura de uma mensagem SOAP



Simples requisição SOAP-RPC

- *Principal aplicação do SOAP, hoje, é RPC sobre HTTP*
  - *Esquema do corpo da mensagem lida com RPC*

```
POST /xmlrpc-bookstore/bookpoint/BookstoreIF HTTP/1.0
Content-Type: text/xml; charset="utf-8"
Content-Length: 585
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
    env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <env:Body>
      <ans1:getPrice xmlns:ans1="http://mybooks.org/wsdl">
         <String_1 xsi:type="xsd:string">2877142566</String_1>
      </ans1:getPrice>
   </env:Body>
</env:Envelope>
```

Parâmetro (ISBN)

Resposta SOAP-RPC

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
SOAPAction: ""
Date: Thu, 08 Aug 2002 01:48:22 GMT
Server: Apache Coyote HTTP/1.1 Connector [1.0]
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:ns0="http://mybooks.org/types"
   env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
    <ans1:getPriceResponse xmlns:ans1="http://mybooks.org/wsdl">
      <result xsi:type="xsd:decimal">19.50</result>
    </ans1:getPriceResponse>
  </env:Body>
</env:Envelope>
```

Resposta (Preço)

Descrição de um serviço RPC: WSDL

- *Para saber usar um Web Service, é preciso*
  - *Saber o que um serviço faz (quais as operações?)*
  - *Como chamar suas operações (parâmetros? tipos?)*
  - *Como encontrar o serviço (onde ele está?)*
- ***W**eb **S**ervices **D**escription **L**anguage*
  - *Documento XML de esquema padrão que contém todas as informações necessárias para que um cliente possa utilizar um Web Service*
  - *Define informações básicas (operações, mapeamentos, tipos, mensagens, serviço) e suporta extensões*
  - *Tem basicamente mesmo papel que linguagens IDL usadas em outros sistemas RPC*
  - *Pode ser usada na geração automática de código*

Interoperabilidade com WSDL

- WSDL serve apenas para descrever interfaces
  - Não serve para ser executada
  - Nenhuma aplicação precisa da WSDL (não faz parte da implementação - é só descrição de interface)
- WSDL pode ser mapeada a linguagens (binding)
  - Mapeamento: tipos de dados, estruturas, etc.
  - Pode-se gerar código de cliente e servidor a partir de WSDL (stubs & skeletons) em tempo de compilação ou execução
- WSDL facilita a interoperabilidade
  - Viabiliza RPC via SOAP
  - Pode-se gerar a parte do cliente em uma plataforma (ex: .NET) e a parte do servidor em outra (ex: J2EE), viabilizando a comunicação entre arquiteturas diferentes.

Exemplo: WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookstoreService"
     targetNamespace="http://mybooks.org/wsdl"
     xmlns:tns="http://mybooks.org/wsdl"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <types>...</types>
    <message name="BookstoreIF_getPrice">
       <part name="String_1" type="xsd:string"/>
    </message>
    <message name="BookstoreIF_getPriceResponse">
       <part name="result" type="xsd:decimal"/>
    </message>
    <portType name="BookstoreIF">
       <operation name="getPrice" parameterOrder="String_1">
          <input message="tns:BookstoreIF_getPrice"/>
          <output message="tns:BookstoreIF_getPriceResponse"/>
       </operation>
    </portType>
    <binding ... > ...</binding>
    <service ... > ... </service>
</definitions>
```

Compare com a mensagem SOAP mostrada anteriormente

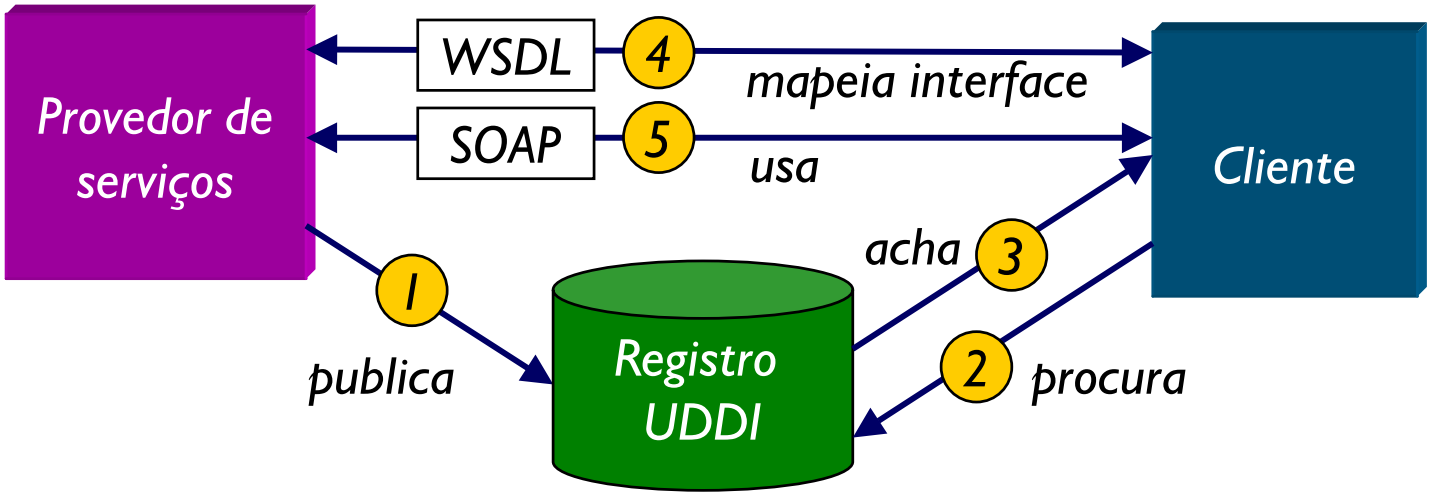
Informa onde está o serviço (endpoint)

Registro e localização do serviço: UDDI

- **U**niversal **D**iscovery and **D**escription **I**ntegration
  - Registro global para Web Services: nuvem UDDI
  - Esquema padrão (XML) para representar firmas, serviços, pontos de acesso dos serviços, relacionamentos, etc.
  - Objetivo é permitir a maior automação no uso dos serviços
  - Registro UDDI acha e devolve URL do WSDL ou serviço
- Registro centralizado permite
  - Independencia de localização
  - Facilidade para pesquisar e utilizar serviços existentes
- Tipos de informações armazenadas em UDDI
  - White pages: busca um serviço pelo nome
  - Yellow pages: busca um serviço por assunto
  - Green pages: busca com base em características técnicas

Web Services: Resumo

- *Arquitetura de serviços usando SOAP, WSDL e UDDI*



- *Comparação com outras soluções de RPC*

	Java RMI	CORBA	RMI / IIOP	Web Services
Registro	RMI Registry	COS Naming	JNDI	UDDI
Descrição de Serviços	Java	OMG IDL	Java	WSDL
Transporte	Java RMI	IIOP	IIOP	SOAP

Tecnologias Java para Web Services

- Java 2 Enterprise Edition (J2EE)
  - Versão 1.3 (atual): já possui todos os recursos necessários para infraestrutura de Web Services (servlets, JSP)
  - Versão 1.4 (2003): integração nativa com Web Services - será mais fácil transformar EJBs e componentes Web em clientes e provedores de Web Services
- Para criar Web Services em Java hoje
  - (1) Java Servlet API 2.3, JSP 1.2, JSTL 1.0
  - (2) Implementações Java de XML, SOAP, UDDI (há várias: IBM WSDL4J, UDDI4J, Apache SOAP, AXIS, Xerces, Xalan)
  - (3) Java XML Pack ("série JAX")

➡ Java Web Services Development Pack = (1) + (3)

Java Web Services Development Pack 1.0

- APIs
  - Processamento XML: **JAXP** 1.1
  - Web Services: **JAX-RPC** 1.0, **JAXM** 1.1, **SAAJ** 1.1, **JAXR** 1.0
  - Aplicações Web: **Servlet API** 2.3, **JSP** 1.2, **JSTL** 1.0
- Implementação de referência
  - Ferramentas de desenvolvimento: Web Deploytool, Compilador JAXRPC (xrpcc), Jakarta Ant, Jakarta Tomcat, Registry Browser e Apache Xindice (banco de dados XML)
  - Serviços de registro UDDI, roteamento SOAP e JAXRPC (implementados como servlets no Tomcat)

xindice
(Zeen-dee-chay)

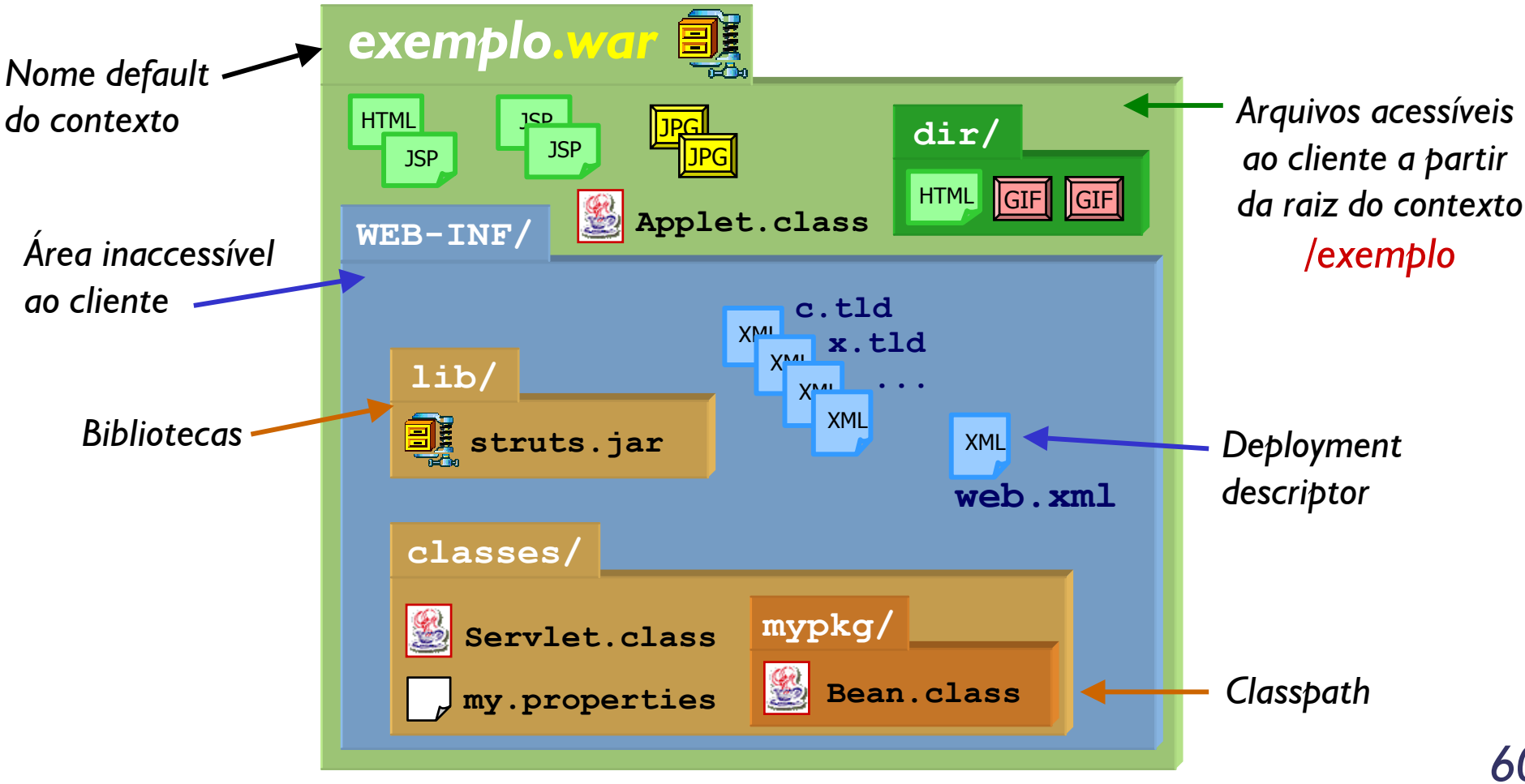
Aplicações Web em Java

- *Web Services podem ser desenvolvidos em Java usando os pacotes javax.servlet.* que permitem criar*
  - *Servlets: componentes capazes de processar requisições HTTP e gerar respostas HTTP*
  - *Páginas JSP: documentos de texto (HTML, XML) que são transformados em servlets na instalação ou execução*
  - *Bibliotecas de tags: implementações que permitem o uso de XML no lugar do código Java em paginas JSP*
- *Deployment é muito simples*
  - *Escreva os servlets / JSPs que implementam Web Services*
  - *Escreva ou gere um deployment descriptor*
  - *Coloque tudo em um arquivo WAR*
  - *Instale o WAR no servidor (ex: copiar p/ pasta webapps/)*

Estrutura de um arquivo WAR

- *Aplicações Web são empacotadas em arquivos WAR para instalação automática em servidores J2EE*

http://servidor.com.br/exemplo

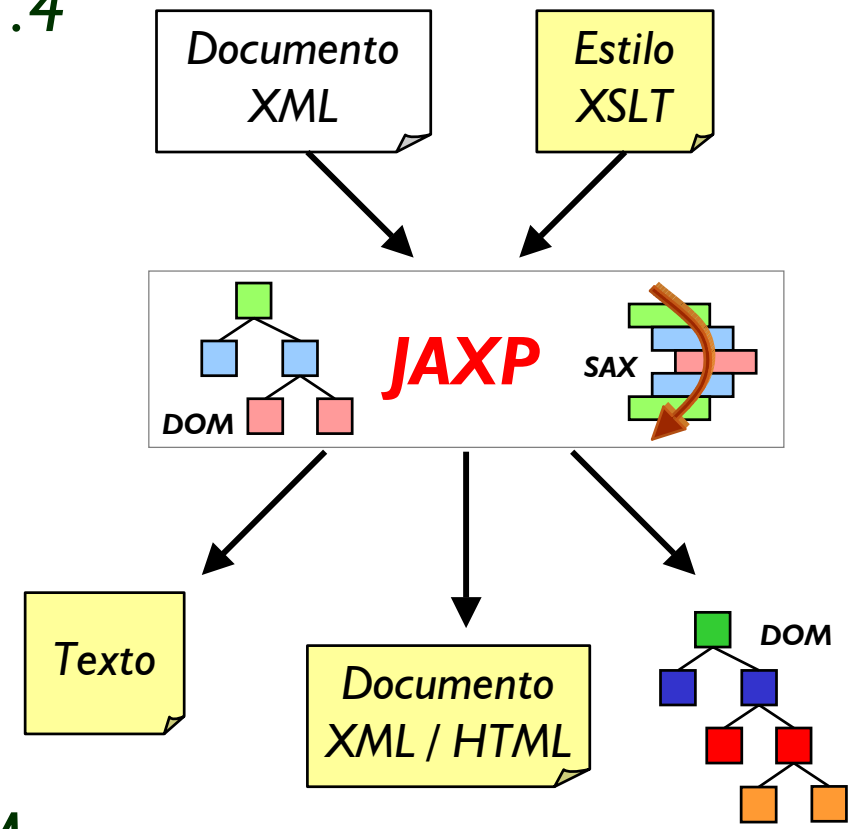


Aplicações XML em Java

- APIs padrão no J2SDK e J2EE
  - **JAXP**: suporte a APIs para processamento XML: DOM , SAX e XSLT
- APIs padrão no Java Web Services Development Pack
  - **JAXM**, **JAX-RPC** e **SAAJ**: suporte a protocolos de comunicação baseados em XML
  - **JAXR**: suporte a sistemas de registro baseados em XML
- Padrões propostos (em desenvolvimento)
  - **JAXB** (JSR-31: XML data binding): suporte à serialização de objetos em XML
  - **JDOM** (JSR-102): outro modelo para processamento XML (que não usa a interface W3C DOM)
  - JSR-181: linguagem de **metadados** para Web Services

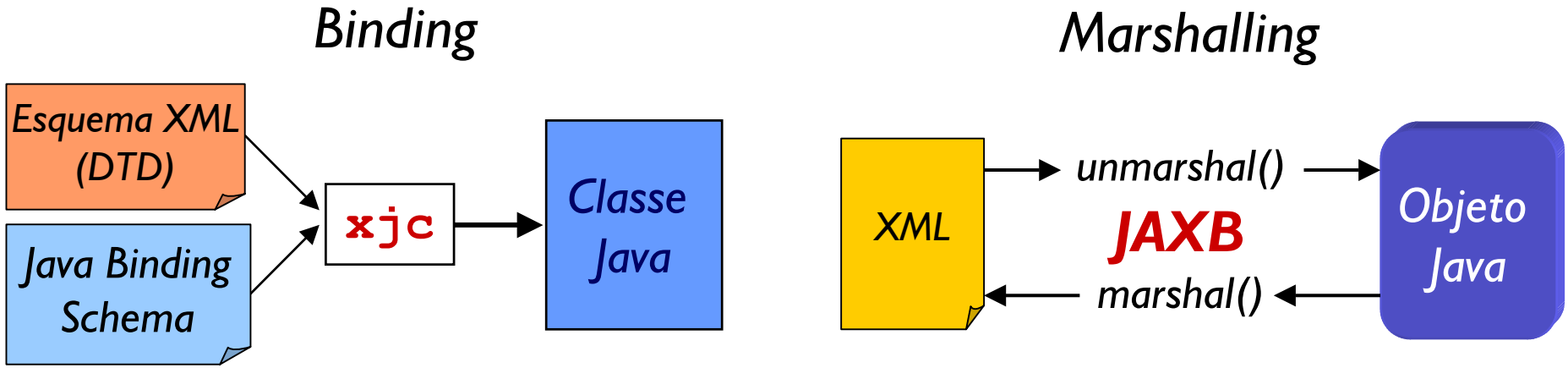
JAXP

- **J**ava **A**PI for **X**ML **P**rocessing
  - Para leitura, criação, manipulação, transformação de XML
  - Parte integrante do J2SDK 1.4
- Pacotes
  - javax.xml.parsers
  - javax.xml.transform.*
  - org.w3c.dom
  - org.w3c.sax.*
- Componentes
  - Parsers para SAX e DOM
  - Implementações em Java das APIs padrão SAX e DOM
  - Implementações Java de API de transformação XSLT



JAXB

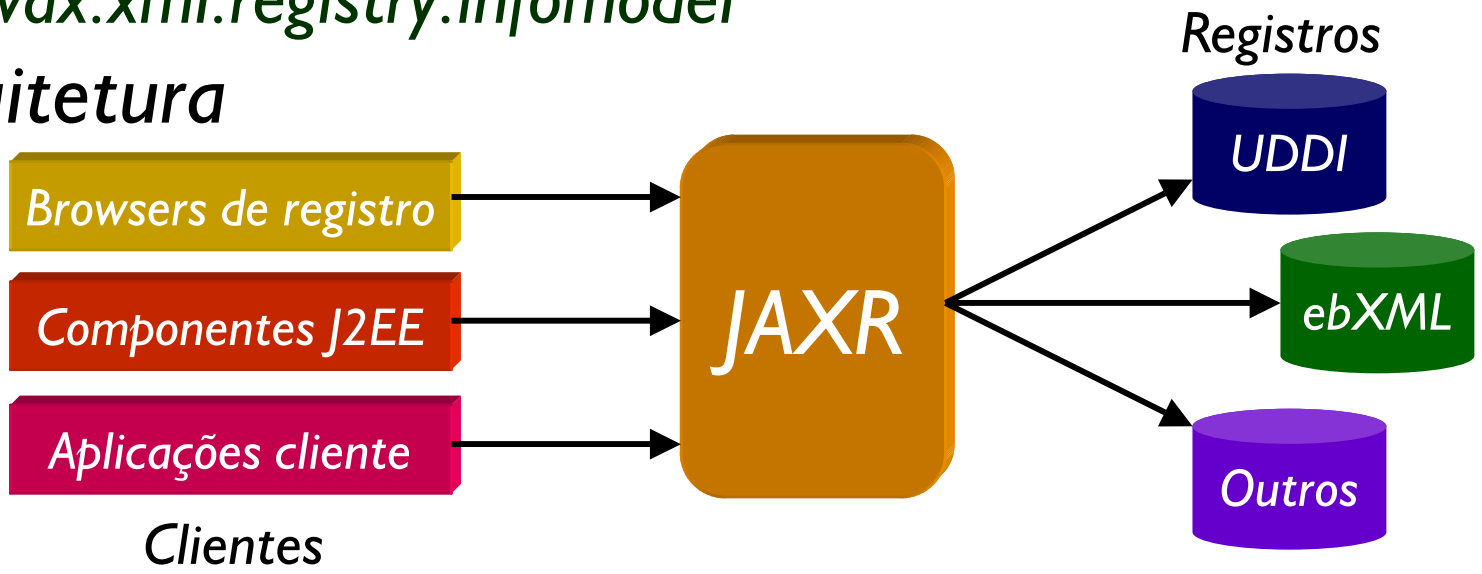
- *Java API for XML Binding (JSR-31)*
  - *Mapeia classes Java a documentos XML*
  - *Permite gerar JavaBeans a partir de esquema XML*
  - *Permite serializar objetos para XML e vice-versa*



- *Pacotes (community review jul-2002)*
  - *javax.xml.bind*
  - *javax.xml.marshall*
- *Em desenvolvimento há 3 anos (29/ago/1999).*

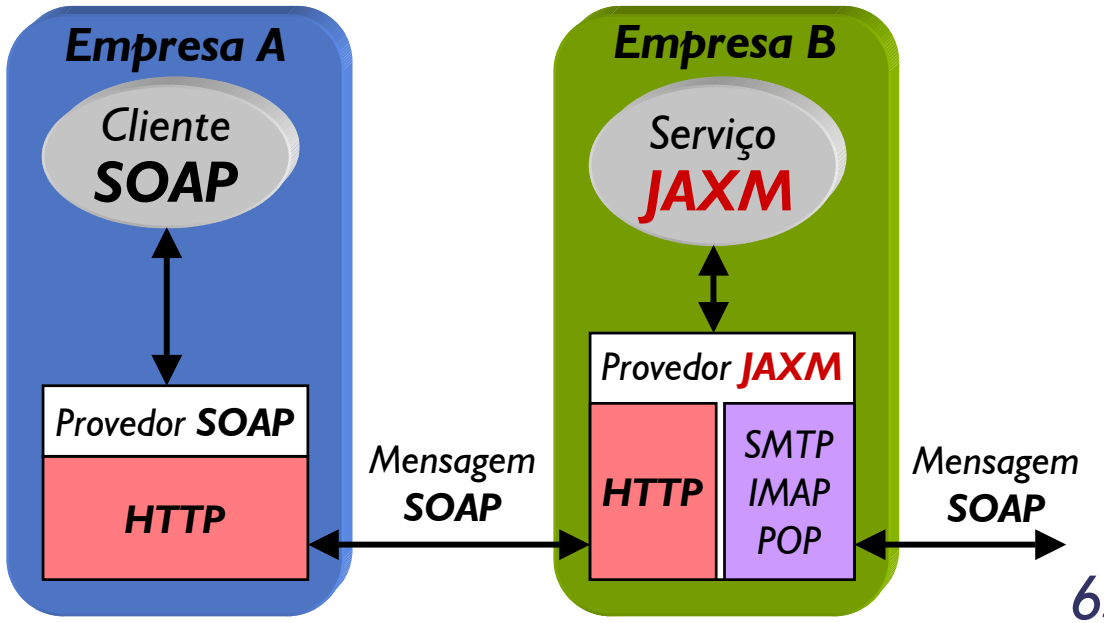
JAXR

- **J**ava **A**PI for **X**ML **R**egistries
  - Oferece acesso uniforme a diferentes sistemas de registro de serviços baseados em XML
  - Possui mapeamentos para UDDI e ebXML
  - Permite a inclusão e pesquisa de organizações, serviços
- Pacotes
  - javax.xml.registry
  - javax.xml.registry.infomodel
- Arquitetura



JAXM (e SAAJ)

- **J**ava **A**PI for **X**ML **M**essaging (e **S**OAP with **A**ttachments **A**PI for **J**ava)
  - Conjunto de APIs para manipular envelopes SOAP e transportá-los sobre HTTP, SMTP ou outros protocolos
  - Suporta comunicação baseada em eventos (mensagens) e baseada em RPC (par de mensagens requisição/resposta)
  - Suporta especificações SOAP 1.1 e SOAP with Attachments
- Pacotes:
  - javax.xml.soap
  - javax.xml.messaging
  - javax.xml.rpc.*



Fonte da ilustração:
JAXM 1.0 specification

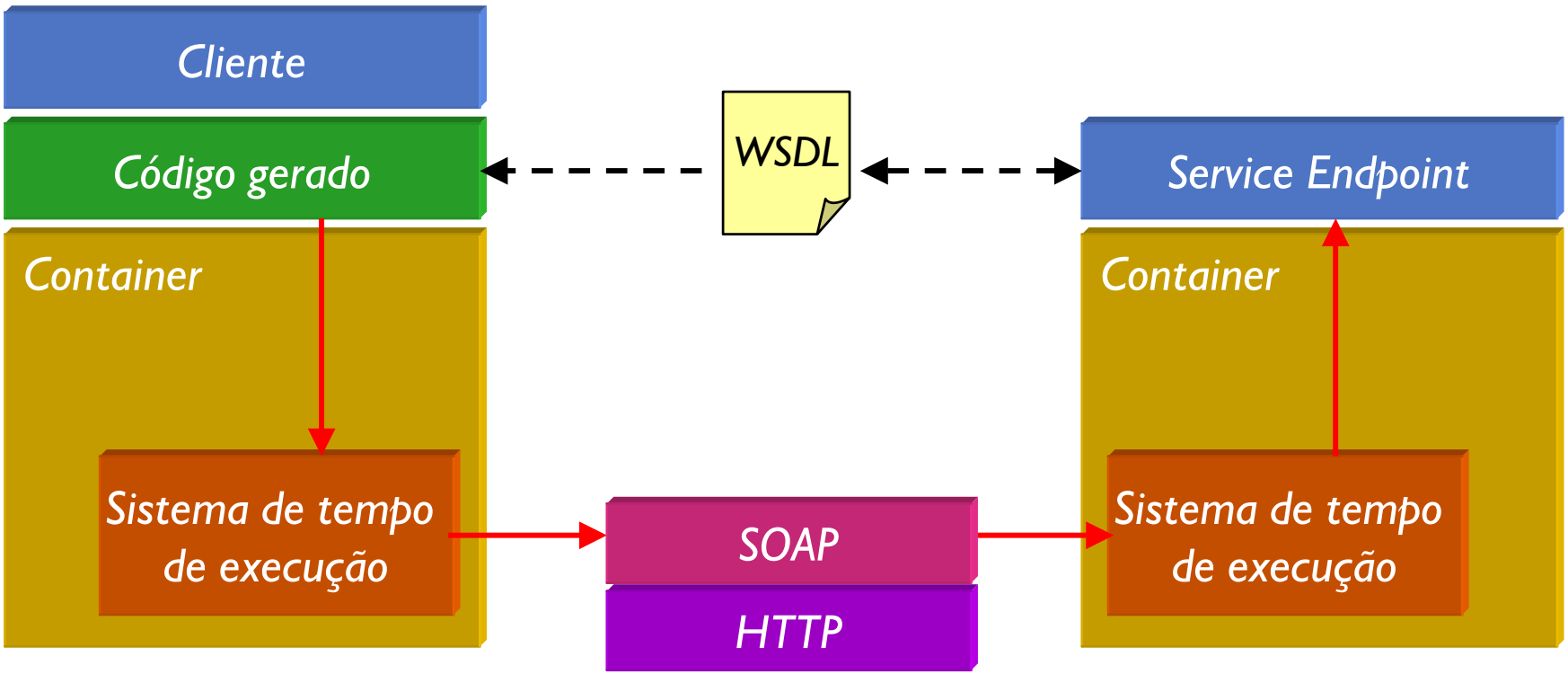
JAX-RPC

- **J**ava **A**PI for **X**ML-Based **R**emote **P**rocedure **C**alls
  - Um tipo de Java RMI sobre SOAP/HTTP
  - Alto nível de abstração permite ignorar envelope SOAP
  - Utiliza WSDL para gerar classes de servidor e cliente
- Pacotes
  - javax.xml.rpc.*
- Desenvolvimento semelhante a RMI (simples e baseado em geração de código e container)
  - Escreve-se RMI, obtém-se SOAP e WSDL
  - Cliente pode obter interface para comunicação com o serviço dinamicamente, em tempo de execução
  - Stubs também podem ser gerados em tempo de compilação para maior performance

JAXM vs. JAX-RPC

- Soluções diferentes para manipular mesmo envelope SOAP
  - JAX-RPC implementa WSDL. JAXM não usa WSDL.
  - JAXM manipula mensagens sem ligar para seu conteúdo
  - JAX-RPC usa WSDL para formato de requisições e respostas
  - JAXM expõe todos os detalhes do envelope; JAX-RPC oculta
  - Tudo o que se faz em JAX-RPC, pode-se fazer com JAXM
  - RPC é mais fácil com JAX-RPC; JAXM é API de baixo nível e pode ser usada tanto para messaging ou RPC
  - Cliente e serviço JAX-RPC rodam em container
- Conclusão
  - Use JAX-RPC para criar aplicações SOAP-RPC com WSDL
  - Use JAXM para messaging ou quando precisar manipular o envelope SOAP diretamente

Arquitetura JAX-RPC



Criação de um Web Service com JAX-RPC (1)

1. Escrever uma interface RMI para o serviço

```
package example.service;

public interface BookstoreIF extends java.rmi.Remote {
   public BigDecimal getPrice(String isbn)
                      throws java.rmi.RemoteException;
}
```

2. Implementar a interface

```
package example.service;

public class BookstoreImpl implements BookstoreIF {

   private BookstoreDB database = DB.getInstance();

   public BigDecimal getPrice(String isbn) {
      return database.selectPrice(isbn);
   }
}
```

Criação de um Web Service com JAX-RPC (2)

3. Escrever arquivo de configuração*

```
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service name="BookstoreService" targetNamespace="http://mybooks.org/wsdl"
                                   typeNamespace="http://mybooks.org/types"
                                   packageName="example.service">
    <interface name="example.service.BookstoreIF"
               servantName="example.service.BookstoreImpl"/>
  </service>
</configuration>
```

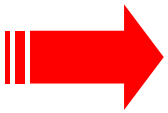
config_rmi.xml

4. Compilar classes e interfaces RMI

```
> javac -d mydir BookstoreIF.java BookstoreImpl.java
```

5. Gerar código do servidor

```
> xrpcc -classpath mydir
        -server -keep
        -d gendir
        config_rmi.xml
```



* Não faz parte da especificação - procedimento pode mudar no futuro

Criação de um Web Service com JAX-RPC (3)

- 6. Criar web deployment descriptor web.xml

```
<web-app>
   <servlet>
      <servlet-name>JAXRPCEndpoint</servlet-name>
      <servlet-class>
        com.sun.xml.rpc.server.http.JAXRPCServlet
      </servlet-class>
      <init-param>
         <param-name>configuration.file</param-name>
         <param-value>
          /WEB-INF/BookstoreService_Config.properties
        </param-value>
      </init-param>
      <load-on-startup>0</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>JAXRPCEndpoint</servlet-name>
      <url-pattern>/bookpoint/*</url-pattern>
   </servlet-mapping>
</web-app>
```

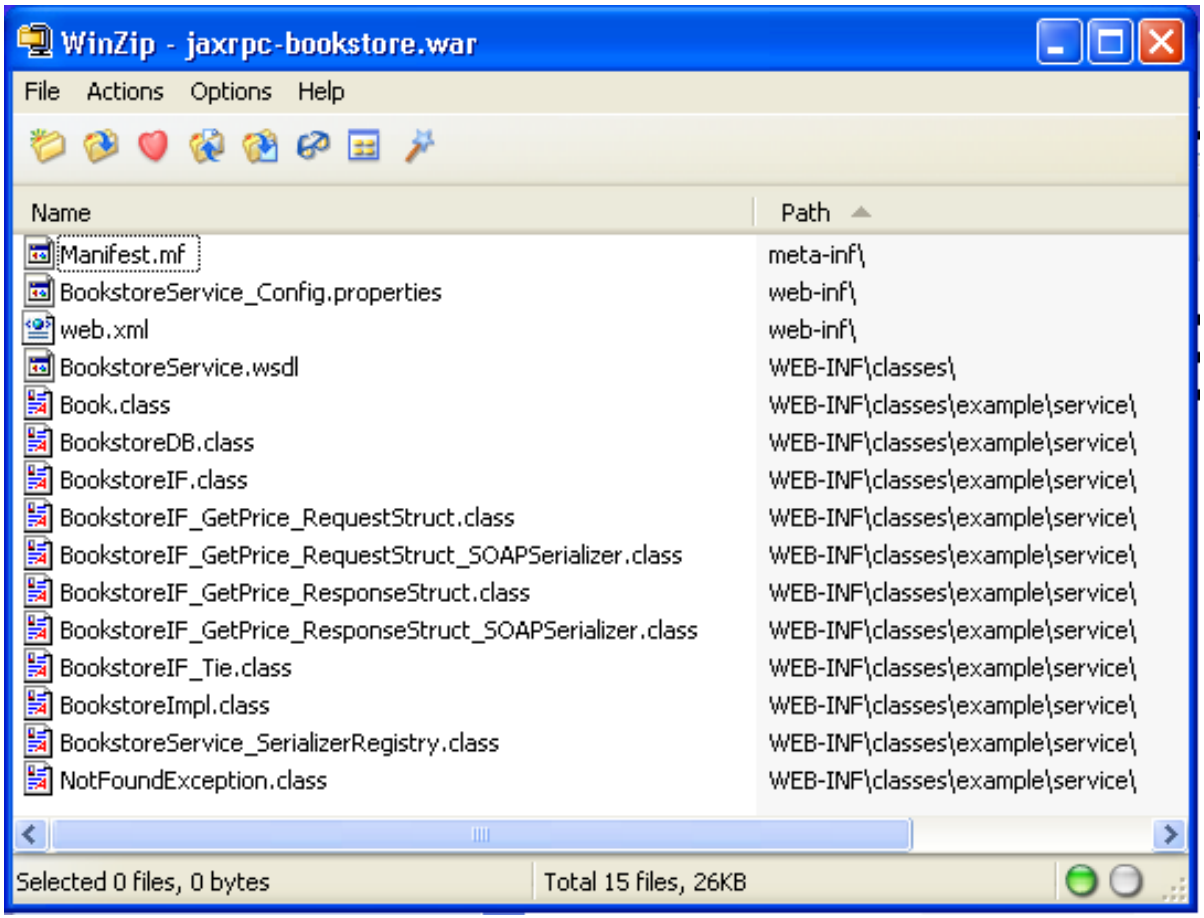
Nosso "container"

Nome do arquivo gerado pelo xrpcc

subcontexto que será o endpoint do serviço

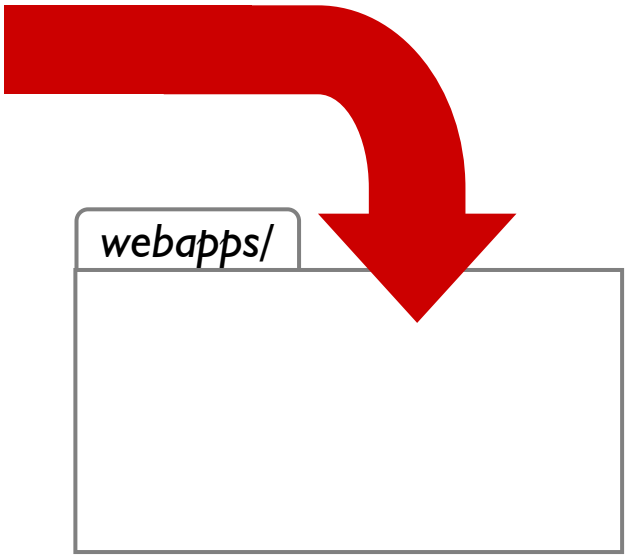
Criação de um Web Service com JAX-RPC (4)

- 7. Colocar tudo em um WAR



jaxrpc-bookstore.war

- 8. Deployment no servidor
  - Copiar arquivo para diretório webapps do Tomcat

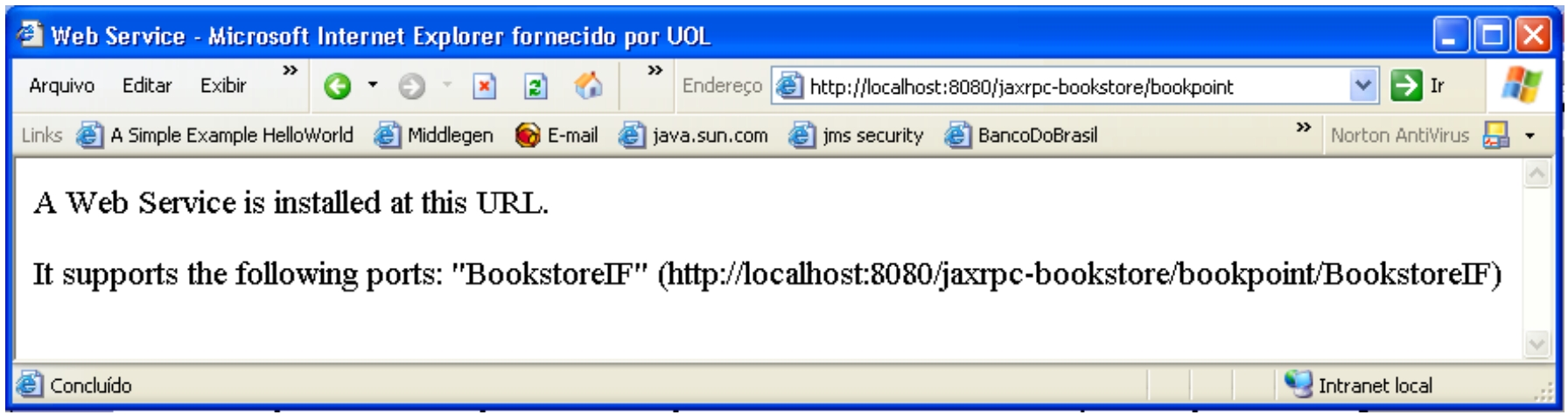


Construção e instalação do serviço com o Ant

- *Script do Ant para compilar as classes RMI, compilá-las com xrpcc, gerar o WSDL, empacotar no WAR e copiar para o diretório webapps/ do Tomcat*

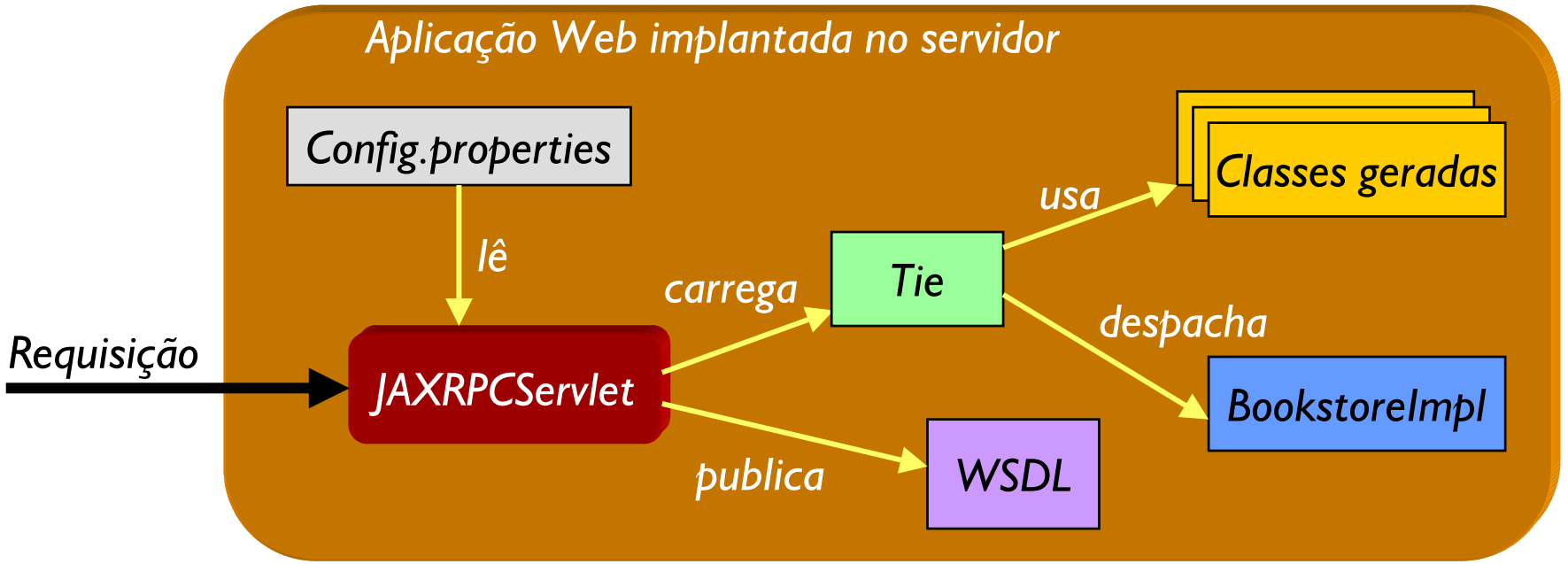
```
> ant BUILD.ALL.and.DEPLOY
```

- *Teste para saber se o serviço está no ar*
  - *Inicie o Tomcat do JWSDP*
  - *Acesse: http://localhost:8080/jaxrpc-bookstore/bookpoint*



Execução

- *O endpoint do serviço na implementação de referência JWSDP 1.0 é um servlet*
  - `com.sun.xml.rpc.server.http.JAXRPCServlet`
    - *Próximas versões (e J2EE 1.4) devem oferecer implementação em stateless session bean*
- *Servlet é ponto de entrada para todas as requisições*

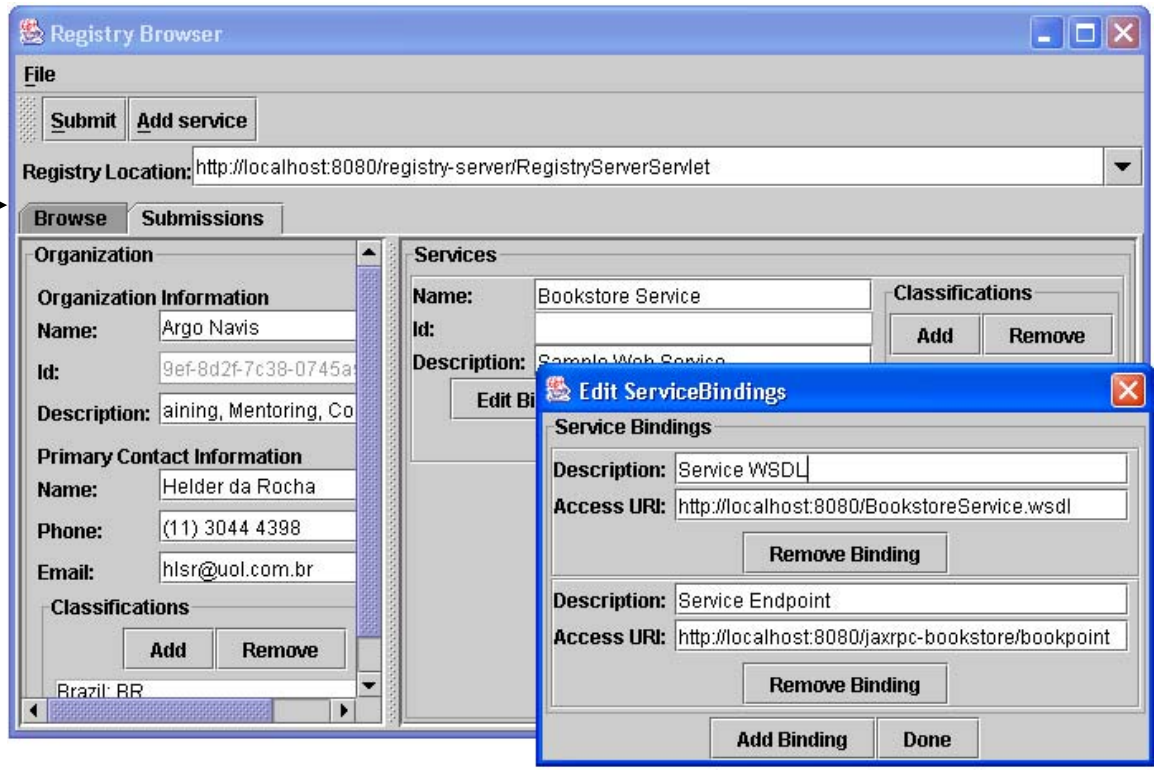


Registro do serviço

- Podemos registrar o nosso Web Service
  - Automaticamente executando um cliente (`ant REGISTER`)
  - Interativamente usando o Registry Browser
- Para usar o servidor UDDI do JWSDP
  - 1. inicie o Xindice
  - 2. inicie o Tomcat

Registry Browser

1. Selecione a localização do servidor (http://localhost/...)
2. Crie uma nova organização
3. Crie novo serviço
4. Em "edit bindings" coloque URLs dos serviços
5. Aperte submit. Use "testuser" como nome e senha

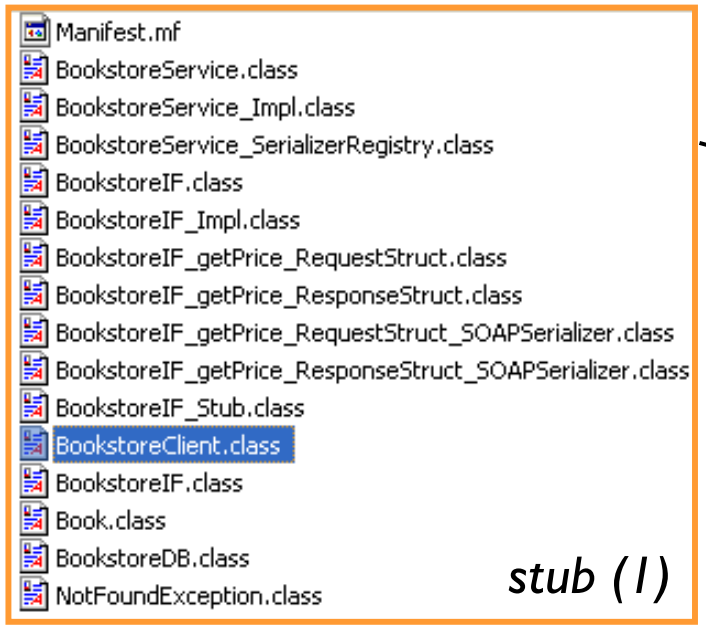


Cliente

- Há três tipos de cliente JAX-RPC:
  - 1. Cliente estático tipo-RMI: usa stubs gerados em tempo de compilação para se comunicar com o servidor e chama métodos do serviço remoto como se fossem locais
  - 2. Cliente WSDL de interface dinâmica (DII): descobre a interface de comunicação em tempo de execução e chama métodos via mecanismo similar a Java reflection
  - 3. Cliente WSDL de interface estática: usa interface Java implementada por stubs gerados em tempo de execução e chama métodos remotos como se fossem locais
- Clientes precisam aderir ao contrato com o Web Service (WSDL) mas podem ser implementados e usados com ou sem WSDL

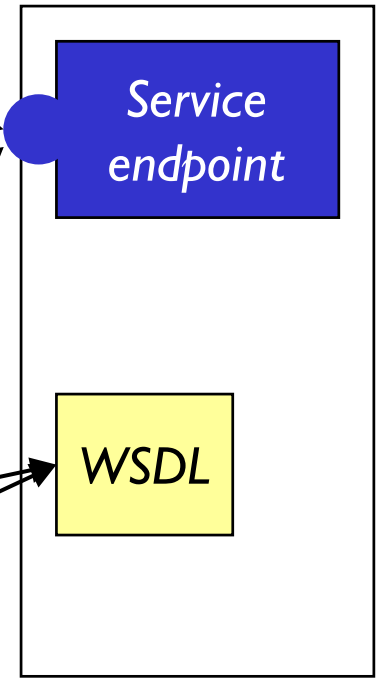
Clientes JAX-RPC

Cliente de implementação estática



+ performance, + acoplamento

Clientes de implementação dinâmica



2. Chama o serviço

1. Obtém informações sobre o serviço

- performance, - acoplamento

Clientes JAX-RPC (detalhes)

- *1) Cliente com stub estático*

```
Stub stub = (Stub)(new BookstoreService_Impl().getBookstoreIFPort());
stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, epointURL);
BookstoreIF proxy = (BookstoreIF)stub;
System.out.println(Price R$ " + proxy.getPrice("2877142566"));
```

- *Cliente com interface dinâmica (Dll)*

```
Service srv = factory.createService(new URL(wsdlURL),
                                    new QName(NS, "BookService"));
Call call = srv.createCall( new QName(NS, "BookstoreIFPort") );
call.setOperationName(new QName(NS, "getPrice"));
BigDecimal pr = (BigDecimal)call.invoke(new String[] {"2877142566"});
System.out.println(Price R$ " + pr);
```

- *Cliente com stub dinâmico (proxy)*

```
Service srv = factory.createService(new URL(wsdlURL),
                                    new QName(NS, "BookService"));
BookstoreIF proxy = (BookstoreIF)
   srv.getPort(new QName(NS, "BookstoreIFPort"),  BookstoreIF.class);
System.out.println(Price R$ " + proxy.getPrice("2877142566") );
```

*Veja código completo em jaxrpcdemo/src/client/**

Execução

- *Para gerar os clientes*
  - *Cliente (1): gere stubs com xrpcc -client e arquivo WSDL (use config_wsdl.xml) e depois compile classe do cliente*
  - *Clientes (2) e (3): apenas compile a classe do cliente*
- *Script do Ant para compilar os três clientes e colocar as classes em um JAR*

```
> ant client.BUILD
```

- *Para rodar o cliente e executar o Web Service*

```
> ant dynamic-client.RUN
Buildfile: build.xml

dynamic-client.RUN:
     [java] ISBN 2877142566. Price R$ 19.50
BUILD SUCCESSFUL
```

Conclusões

- Nesta palestra apresentamos a arquitetura de Web Services, suas tecnologias fundamentais SOAP, WSDL e UDDI e as APIs Java que as implementam.
- Java oferece APIs que permitem desde a manipulação direta de XML (DOM e SAX) até a criação de Web Services sem contato com XML (JAX-RPC)
- JAX-RPC é a forma mais fácil e rápida de criar Web Services em Java
- Serviços desenvolvidos em JAX-RPC poderão ser acessados de aplicações .NET e vice-versa.
  - Web Services viabilizam a integração de serviços entre plataformas diferentes: interoperabilidade!

Fontes

[1] JSR-101 Expert Group. Java™ API for XML-based RPC: JAX-RPC 1.0 Specification. Java Community Process: www.jcp.org.

[2] Sun Microsystems. Java™ Web Services Tutorial. java.sun.com/webservices/. Coleção de tutoriais sobre XML, JSP, servlets, Tomcat, SOAP, JAX-RPC, JAXM, etc.

[3] JSR-109 Expert Group. Web Services for J2EE 1.0 (Public Draft 15/04/2002). Java Community Process: www.jcp.org. Descreve o suporte a Web Services em J2EE 1.3

[4] Nicholas Kassem et al. (JSR-67). Java™ API for XML Messaging (JAXM) e Soap with Attachments API for Java 1.1. java.sun.com. Modelo de programação de baixo nível (lida diretamente com SOAP enquanto JAX-RPC esconde) e mais abrangente.

[5] Roberto Chinnici. Implementing Web Services with the Java™ Web Services Development Pack. JavaONE Session 1777. java.sun.com/javaone. Apresentação que oferece uma visão garal de JAX-RPC e o Web Services Development Pack da Sun.

[6] Brett McLaughlin. Java & XML 2nd. Edition. O'Reilly and Associates, 2001. Explora as APIs Java para XML e oferece uma introdução à programação de WebServices em Java

[7] Ethan Cerami. Web Services Essentials. O'Reilly, Fev 2002. XML-RPC, SOAP, UDDI e WSDL são explorados de forma didática e exemplos são implementados em Java usando ferramentas open-source.

[8] W3C Web Services Activity. http://www.w3.org/2002/ws/. Página que dá acesso aos grupos de trabalho que desenvolvem especificações de SOAP (XMLP), WSDL e Arquitetura

Fontes

[9] *Apache XML Project*. xml.apache.org. *Duas implementações de SOAP e uma implementação de XML-RPC em Java.*

[10] *IBM Developerworks Open Source Projects*. http://www-124.ibm.com/. *Implementações UDDI4J e WSDL4J.*

[11] Al Saganich. *Java and Web Services Primer*. O'Reilly Network 2001. http://www.onjava.com/pub/a/onjava/2001/08/07/webservices.html. *Ótimo tutorial sobre Web Services.*

[12] Al Saganich. *Hangin' with the JAX Pack. Part 1: JAXP and JAXB, Part 2: JAXM, Part 3: Registries (JAXR), Part 4: JAX-RPC*. O'Reilly Network 2001-2002. http://www.onjava.com/pub/a/onjava/2001/11/07/jax.html *Esta série de quatro artigos publicados entre nov/2001 e abr/2002 é talvez o melhor ponto de partida para quem desejar aprender a usar as APIs Java para Web Services.*

[13] David Chappell, Tyler Jewel. *Java Web Services*. O'Reilly and Associates, Mar 2002. *Explora implementações Java de Apache SOAP, WSDL e UDDI em Java. Tem um capítulo dedicado às APIs do JWSDP.*

[14] Al Saganich. *JSR-109 Web Services inside of J2EE Apps*. O'Reilly Network, Aug 2002. http://www.onjava.com/pub/a/onjava/2002/08/07/j2eewebsvs.html *Mostra um resumo da proposta do JSR-109, que prevê a integração J2EE-Web Services.*

helder@argonavis.com.br

www.argonavis.com.br

Palestra: Como Implementar Web Services em Java
COMDEX 2002, São Paulo
Curso: XML 100 - Introdução a XML
© 2001, 2002, Helder da Rocha